

Rudolf Fleischer
Gerhard Trippen (Eds.)

LNCS 3341

Algorithms and Computation

15th International Symposium, ISAAC 2004
Hong Kong, China, December 2004
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

New York University, NY, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Rudolf Fleischer Gerhard Trippen (Eds.)

Algorithms and Computation

15th International Symposium, ISAAC 2004
Hong Kong, China, December 20-22, 2004
Proceedings

Volume Editors

Rudolf Fleischer
Fudan University
Department of Computer Science and Engineering
220 Handan Road, 200433 Shanghai, China
E-mail: fleischer@acm.org

Gerhard Trippen
The Hong Kong University of Science and Technology
Department of Computer Science
Clear Water Bay, Kowloon, Hong Kong, China
E-mail: trippen@cs.ust.hk

Library of Congress Control Number: 2004116722

CR Subject Classification (1998): F.2, C.2, G.2-3, I.3.5, F.1

ISSN 0302-9743

ISBN 3-540-24131-0 Springer Berlin Heidelberg New York

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

Springer is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11369226 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 15th Annual International Symposium on Algorithms and Computation (ISAAC 2004), held in Hong Kong, 20–22 December, 2004. In the past, it has been held in Tokyo (1990), Taipei (1991), Nagoya (1992), Hong Kong (1993), Beijing (1994), Cairns (1995), Osaka (1996), Singapore (1997), Taejon (1998), Chennai (1999), Taipei (2000), Christchurch (2001), Vancouver (2002), and Kyoto (2003).

ISAAC is an annual international symposium that covers a wide range of topics, namely algorithms and computation. The main purpose of the symposium is to provide a forum for researchers working in the active research community of algorithms and the theory of computation to present and exchange new ideas.

In response to our call for papers we received 226 submissions. The task of selecting the papers in this volume was done by our program committee and other referees. After a thorough review process the committee selected 76 papers, the decisions being based on originality and relevance to the field of algorithms and computation. We hope all accepted papers will eventually appear in scientific journals in a more polished form. Two special issues, one of *Algorithmica* and one of the *International Journal of Computational Geometry and Applications*, with selected papers from ISAAC 2004 are in preparation.

The best student paper award will be given for “Geometric optimization problems over sliding windows” by Bashir S. Sadjad and Timothy M. Chan from the University of Waterloo. Two eminent invited speakers, Prof. Erik D. Demaine, MIT, and Prof. David M. Mount, University of Maryland, also contributed to this volume.

It is impossible to organize such a conference without the help of many individuals. We would like to express our appreciation to the authors of the submitted papers, and to the program committee members and external referees for their excellent work. We also would like to thank the organizing committee, most notably Gerhard Trippen, for their tremendous work in making ISAAC 2004 a successful conference. Finally, we thank our sponsors, the Croucher Foundation and the K.C. Wong Education Foundation, for their assistance and support.

October 2004

Rudolf Fleischer
Gerhard Trippen

Organization

Program Committee

Lars Arge, S
Michael Bender, S S
Therese Biedl,
Adam Buchsbaum, & S
Siu-Wing Cheng, S
Xiaotie Deng,
Rudolf Fleischer, (Chair)
Leszek Gasieniec,
Seokhee Hong, S
Wen-Lian Hsu, S
Tak-Wah Lam,
Guojun Li, S
Tomomi Matsui,
Christoph Meinel,
Friedhelm Meyer auf der Heide,
Mike Paterson,
Knut Reinert,
Jörg-Rüdiger Sack,
Hiroki Shizuya,
Jack Snoeyink, S
Roberto Solis-Oba,
Ulrike Stege,
John Watrous,
Gerhard Woeginger,
Hsu-Chun Yen,

Organizing Committee

Sunil Arya, S
Leizhen Cai,
Siu-Wing Cheng, S
Francis Chin,
Xiaotie Deng,
Rudolf Fleischer (chair),
Mordecai Golin, S
Tak-Wah Lam,
Gerhard Trippen, S
Derick Wood, S
Frances Yao,
Joseph Zhen Zhou, S

Sponsors

The Croucher Foundation, Hong Kong, China

The K.C. Wong Education Foundation, Hong Kong, China

Referees

Pankaj Agarwal
 Heekap Ahn
 Amihood Amir
 Nikhil Bansal
 Marcin Bienkowski
 Johannes Blömer
 Carsten Böke
 Franz J. Brandenburg
 Gruia Calinescu
 H.L. Chan
 W.T. Chan
 Bernard Chazelle
 Bogdan Chelebus
 Ning Chen
 Xi Chen
 Yong Chen
 Tae-nam Cho
 Graham Cormode
 Valentina Damerow
 Bhaskar DasGupta
 Benjamin Doerr
 Andreas Döring
 Adrian Dumitrescu
 Peter Eades
 John Ellis
 Leah Epstein
 Thomas Erlebach
 Hongbin Fan
 Qizhi Fang
 Martin Farach-Colton
 Arash Farzan
 Fedor Fomin
 Gereon Frahling
 P.Y. Fung
 Luisa Gargano
 Mohammad GhasemZadeh
 Clemens Gröpl
 Qianping Gu
 Joachim Gudmundsson
 Arvind Gupta
 Torben Hagerup
 Mikael Hammar
 Herman Haverkort
 Chin-Wen Ho
 W.K. Hon
 Tsan-Sheng Hsu
 Ji Hu
 Sha Huang
 Wanjun Huang
 John Iacono
 Martin Isenburg
 Michael Jacobson
 Xiaohua Jia
 David Johnson
 Valerie King
 Ralf Klasing
 Bettina Klinz
 Volker Klotz
 Ming-Tat Ko
 Jochen Konemann
 Yehuda Koren
 Miroslav Korzeniowski
 Dariusz Kowalski
 Miroslaw Kowaluk
 Piotr Krysta
 Eduardo Laber
 Gad M. Landau
 D.T. Lee
 Jianping Li
 Shuguang Li
 Weifa Liang
 Minming Liless
 Yuanxin (Leo) Liu
 Chi-Jen Lu
 Hsueh-I Lu
 Anna Lubiw
 Veli Mäkinen
 Andrea Mantler
 Alexander Martin
 Monaldo Mastrolilli
 Wendy Myrvold
 Nikola Nikolov
 Naomi Nishimura
 Chong-Dae Park
 Jung-Heum Park
 Kunsoo Park

Andrzej Pelc
David Peleg
Xingqin Qi
Prabakhar Ragde
Manuel Rode
Günter Rote
Stefan Rührup
Frank Ruskey
David Sankoff
Christian Scheideler
Volker Schillings
Allan Scott
Jiri Sgall
Chan-Su Shin
Michiel Smid
Christian Sohler
Frits Spijksma
Venkatesh Srinivasan
Xiaoming Sun
Ting-Yi Sung
Don E. Taylor
Gerard Tel
Mikkel Thorup
H.F. Ting
I-Ming Tsai
George Tzanetakis
Miguel Vargias Martin
Suresh Venkatasubramanian

Tomas Vinar
Berthold Vöcking
Klaus Volbert
Lutz Vorwerk
Bow-Yaw Wang
Da-Wei Wang
Lusheng Wang
Wei Wang
Xiaoli Wang
Rolf Wanka
Ingo Wegener
Alexander Wolff
Duncan Wong
Prudence Wong
David R. Wood
Kuen-Pin Wu
Kui Wu
Qin Xin
Yinfeng Xu
S.M. Yiu
Jinjiang Yuan
Michael Zastre
Jing Zhang
Shaoqiang Zhang
Yunlei Zhao
Binhai Zhu
Uri Zwick

Table of Contents

Puzzles, Art, and Magic with Algorithms	1
The ABCs of AVDs: Geometric Retrieval Made Simple	2
Pareto Optimality in House Allocation Problems	3
Property-Preserving Data Reconstruction	16
On the Monotone Circuit Complexity of Quadratic Boolean Functions	28
Generalized Function Matching	41
Approximate Distance Oracles for Graphs with Dense Clusters	53
Multicriteria Global Minimum Cuts	65
Polyline Fitting of Planar Points Under Min-sum Criteria	77
A Generalization of Magic Squares with Applications to Digital Halftoning	89
Voronoi Diagrams with a Transportation Network on the Euclidean Plane	101

Structural Alignment of Two RNA Sequences with Lagrangian Relaxation	113
Poly-APX- and PTAS-Completeness in Standard and Differential Approximation	124
Efficient Algorithms for k Maximum Sums	137
Equipartitions of Measures by 2-Fans	149
Augmenting the Edge-Connectivity of a Spider Tree	159
On Nash Equilibria for Multicast Transmissions in Ad-Hoc Wireless Networks	172
Structural Similarity in Graphs (A Relaxation Approach for Role Assignment)	184
Flexibility of Steiner Trees in Uniform Orientation Metrics	196
Random Access to Advice Strings and Collapsing Results	209
Bounding the Payment of Approximate Truthful Mechanisms	221
The Polymatroid Steiner Problems	234
Geometric Optimization Problems Over Sliding Windows	246
On-Line Windows Scheduling of Temporary Items	259

Generalized Geometric Approaches for Leaf Sequencing Problems in Radiation Therapy	271
An Efficient Exact Algorithm for the Minimum Ultrametric Tree Problem	282
On the Range Maximum-Sum Segment Query Problem	294
An Efficient Algorithm for Finding Maximum Cycle Packings in Reducible Flow Graphs	306
Efficient Job Scheduling Algorithms with Multi-type Contentions	318
Superimposing Voronoi Complexes for Shape Deformation	330
On Partial Lifting and the Elliptic Curve Discrete Logarithm Problem	342
Guarding Art Galleries by Guarding Witnesses	352
On p -Norm Based Locality Measures of Space-Filling Curves	364
Composability of Infinite-State Activity Automata	377
Error Compensation in Leaf Root Problems	389
On Compact and Efficient Routing in Certain Graph Classes	402
Randomized Insertion and Deletion in Point Quad Trees	415
Diagnosis in the Presence of Intermittent Faults	427

Three-Round Adaptive Diagnosis in Binary n -Cubes	442
Fast Algorithms for Comparison of Similar Unordered Trees	452
GCD of Random Linear Forms	464
On the Hardness and Easiness of Random 4-SAT Formulas	470
Minimum Common String Partition Problem: Hardness and Approximations	484
On the Complexity of Network Synchronization	496
Counting Spanning Trees and Other Structures in Non-constant-jump Circulant Graphs	508
Adaptive Spatial Partitioning for Multidimensional Data Streams	522
Paired Pointset Traversal	534
Approximated Two Choices in Randomized Load Balancing	545
Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting	558
Local Gapped Subforest Alignment and Its Application in Finding RNA Structural Motifs	569
The Maximum Agreement of Two Nested Phylogenetic Networks	581

Sequences of Radius k : How to Fetch Many Huge Objects into Small Memory for Pairwise Computations	594
New Bounds on Map Labeling with Circular Labels	606
Optimal Buffer Management via Resource Augmentation	618
Oriented Paths in Mixed Graphs	629
Polynomial Deterministic Rendezvous in Arbitrary Graphs	644
Distributions of Points and Large Quadrangles	657
Cutting Out Polygons with Lines and Rays	669
Advantages of Backward Searching - Efficient Secondary Memory and Distributed Implementation of Compressed Suffix Arrays	681
Inner Rectangular Drawings of Plane Graphs	693
Approximating the Minmax Subtree Cover Problem in a Cactus	705
Boundary-Optimal Triangulation Flooding	717
Exact Computation of Polynomial Zeros Expressible by Square Roots	729
Many-to-many Disjoint Path Covers in a Graph with Faulty Elements	742
An $O(n \log n)$ -Time Algorithm for the Maximum Constrained Agreement Subtree Problem for Binary Trees	754

Planning the Transportation of Multiple Commodities in Bidirectional Pipeline Networks	766
Efficient Algorithms for the Hotlink Assignment Problem: The Worst Case Search	778
Dynamic Tree Cross Products	793
Spanners, Weak Spanners, and Power Spanners for Wireless Networks	805
Techniques for Indexing and Querying Temporal Observations for a Collection of Objects	822
Approximation Algorithms for the Consecutive Ones Submatrix Problem on Sparse Matrices	835
The Two-Guard Problem Revisited and Its Generalization	847
Canonical Data Structure for Interval Probe Graphs	859
Efficient Algorithms for the Longest Path Problem	871
Randomized Algorithms for Motif Detection	884
Weighted Coloring on Planar, Bipartite and Split Graphs: Complexity and Improved Approximation	896
Sweeping Graphs with Large Clique Number	908

A Slightly Improved Sub-Cubic Algorithm for the All Pairs Shortest Paths Problem with Real Edge Lengths 921

Author Index 933

Puzzles, Art, and Magic with Algorithms

Erik D. Demaine

MIT Computer Science and Artificial Intelligence Laboratory,
32 Vassar Street, Cambridge, MA 02139, USA
edemaine@mit.edu

Solving and designing puzzles, creating sculpture and architecture, and inventing magic tricks all lead to fun and interesting algorithmic problems. This talk describes some of our explorations into these areas.

Solving a puzzle is like solving a research problem. Both require the right cleverness to see the problem from the right angle, and then explore that idea until you find a solution. The main difference is that the puzzle poser usually guarantees that the puzzle is solvable. Puzzles also lead to the meta-puzzle of how to design algorithms that can design families of puzzles.

Elegant algorithms are beautiful. A special treat is when that beauty translates visually. Sometimes this is by design, when you develop an algorithm to compose artwork within a particular family. Other times the visual beauty of an algorithm just appears, without anticipation.

Mathematics is the basis for many magic tricks, particularly “self-working” tricks. One of the key people at the intersection of mathematics and magic is Martin Gardner, whose work has inspired several of the results described here. Algorithmically, our goal is to automatically design families of magic tricks.

This is joint work with Martin Demaine and several others.

The ABCs of AVDs: Geometric Retrieval Made Simple

David M. Mount

Department of Computer Science and
Institute for Advanced Computer Studies,
University of Maryland,
College Park, Maryland 20742, USA
`mount@cs.umd.edu`

One of the best known structures in computational geometry is the Voronoi diagram. Given a set of n points in space, called sites, this is a subdivision that partitions space into n convex cells according which site is closest. The Voronoi diagram and its dual, the Delaunay triangulation are among the most fundamental proximity structures known, and have numerous uses in science. For example, applying point location to the subdivision defined by a Voronoi diagram answers nearest neighbor queries. Although they are popular in low dimensional spaces, there are various reasons why Voronoi diagrams are rarely used higher dimensions. First, the combinatorial complexity of the diagram grows rapidly, as fast as $\Omega(n^{d/2})$ in d -dimensional space. Second, the diagram does not admit a straightforward search structure in higher dimensions.

The AVD, or Approximate Voronoi Diagram, was first proposed by Har-Peled as a simple and easily implemented alternative to the Voronoi diagram for answering nearest neighbor queries. Given a user-supplied approximation bound, $\epsilon > 0$, this structure is based on a simple hierarchical decomposition of space into rectangular cells, much like a quadtree. Given this structure, ϵ -approximate nearest neighbor queries can be answered by simply locating the cell that contains the query point, followed perhaps by a small number of distance calculations. If constructed properly, this remarkably simple structure provides the most efficient approach known for answering approximate nearest neighbor queries in spaces of constant dimensionality.

Recent research has shown that the AVD can be adapted to answer other geometric retrieval problems. In this talk we will discuss a number of aspects of the AVD and its applications. This includes its origins, improved algorithms and analysis of its construction and space complexity, generalizations to approximate k -nearest neighbor searching, metric range searching, and other generalizations.

This is based on joint work with Sunil Arya and Theocharis Malamatos.

Pareto Optimality in House Allocation Problems

David J. Abraham^{1,*}, Katarína Cechlárová², David F. Manlove^{3,**},
and Kurt Mehlhorn⁴

¹ Computer Science Department, Carnegie-Mellon University, 5000 Forbes Ave,
Pittsburgh PA 15213-3890, USA

dabraham@cs.cmu.edu

² Institute of Mathematics, P.J. Šafárik University in Košice, Faculty of Science,
Jesenná 5, 040 01 Košice, Slovakia

cechlarova@science.upjs.sk

³ Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK

davidm@dcs.gla.ac.uk

⁴ Max-Planck-Institut für Informatik, Stuhlsatztenhausweg 85,

66123 Saarbrücken, Germany

mehlhorn@mpi-sb.mpg.de

Abstract. We study Pareto optimal matchings in the context of house allocation problems. We present an $O(\sqrt{nm})$ algorithm, based on Gale's Top Trading Cycles Method, for finding a maximum cardinality Pareto optimal matching, where n is the number of agents and m is the total length of the preference lists. By contrast, we show that the problem of finding a minimum cardinality Pareto optimal matching is NP-hard, though approximable within a factor of 2. We then show that there exist Pareto optimal matchings of all sizes between a minimum and maximum cardinality Pareto optimal matching. Finally, we introduce the concept of a signature, which allows us to give a characterization, checkable in linear time, of instances that admit a unique Pareto optimal matching.

1 Introduction

We study the problem of allocating a set H of heterogeneous indivisible goods among a set A of agents [14, 8, 3, 4]. We assume that each agent $a \in A$ ranks in order of preference a subset of H (the *desired* goods for a) and that monetary compensations are not possible. In the literature the situation in which each agent initially owns one good is known as a *house allocation problem* [14, 12, 11]. When there are no initial property rights, we obtain the *house allocation with no initial property rights* [8, 16, 1]. A mixed model, in which a subset of agents initially owns a good has also been studied [2]. Yuan [15] describes a large-scale application of these problems in the allocation of families to government-subsidized housing in China.

* Work done whilst at Department of Computing Science, University of Glasgow, and Max-Planck-Institut für Informatik.

** Supported by grant GR/R84597/01 from the Engineering and Physical Sciences Research Council and RSE/Scottish Executive Personal Research Fellowship.

Following convention we refer to the elements of H as h_1, \dots, h_m , though the class of problems under consideration could equally be formulated in terms of allocating graduates to trainee positions, professors to offices, clients to servers, etc. For ease of exposition we begin by assuming that there are no initial property rights, though we later show how to take account of such a situation.

Given such a problem instance, the task is to construct a (A, H) matching, i.e. a subset M of $A \times H$ such that $(a, h) \in M$ implies that a finds h acceptable, each agent is assigned to at most one house and vice versa. Furthermore one seeks a matching that is *Pareto optimal* in a precise sense, taking into account the agents' preferences. Various notions of optimality have been considered in the literature, but a criterion that has received much attention, particularly from economists, is *maximum cardinality Pareto optimality*. A matching M is Pareto optimal if there is no other matching M' such that no agent is worse off in M' than in M , whilst some agent is better off in M' than in M . For example, a matching M is not Pareto optimal if two agents could improve by swapping the houses that they are assigned to in M .

There is a straightforward greedy algorithm, which we denote by Greedy-POM, for finding a Pareto optimal matching [1]: consider each agent a in turn, giving a his/her most-preferred vacant house (assuming such a house exists). This algorithm is also known as a *serial dictatorship* mechanism [1]. Roth and Sotomayor [13–Example 4.3] remark that a similar mechanism is used by the United States Naval Academy in order to match graduating students to their first posts as Naval Officers (in this context however, the algorithm considers each student in non-decreasing order of graduation results). However one may construct an example instance (see Section 2 for further details) in which Pareto optimal matchings may have different cardinalities and Greedy-POM could fail to produce a Pareto optimal matching of maximum size. Yet in many applications, one wishes to match as many agents as possible.

Stronger notions of optimality have been considered in the literature. For example a matching M is *rank-maximal* [10] if, in M , the maximum number of agents are matched to their first-choice house, and subject to this condition, the maximum number of agents are matched to their second-choice house, and so on. Irving et al. [10] describe two algorithms for finding a rank-maximal matching, with complexities $O(\min\{n + C, C\sqrt{n}\}m)$ and $O(Cnm)$, where $n = |A| + |H|$, m is the total length of the preference lists and C is the maximum k such that some agent is assigned to his/her k th-choice house in the constructed matching. Clearly a rank-maximal matching is Pareto optimal, however a rank-maximal matching need not be a maximum cardinality Pareto optimal matching (henceforth a maximum Pareto optimal matching). Alternatively, one may consider a *maximum cardinality maximum utility* matching M , in which we maximise $\sum_{(a,h) \in M} u_{a,h}$ over all maximum cardinality matchings, where $u_{a,h}$ indicates the utility of house h being allocated to agent a . If one defines $u_{a,h} = l - \text{rank}_{a,h}$, where $\text{rank}_{a,h}$ is the rank of house h in agent a 's preference list and l is the maximum length of an agent's list, then a maximum cardinality maximum utility matching is in turn a maximum Pareto optimal matching. Since all utilities are integral, a maximum cardinality maximum utility matching may be found

in $O(\sqrt{nm} \log n)$ time [5]. However if one only requires to find a maximum cardinality matching that satisfies the weaker condition of being Pareto optimal, it is of interest to consider whether faster algorithms exist.

The next two sections of this paper work towards answering this question. In Section 2 we give a formal definition of the problem model, and present necessary and sufficient conditions for a matching to be Pareto optimal. In Section 3 we use these conditions as the basis for an $O(\sqrt{nm})$ algorithm for finding a maximum Pareto optimal matching. This algorithm extends the Top Trading Cycles Method due to Gale [14], which has been the focus of much attention, particularly in the game theory and economics literature [14, 12, 11, 15, 2]. We then show that any improvement to the complexity of our algorithm would imply an improved algorithm for finding a maximum matching in a bipartite graph. We also demonstrate how to modify our algorithm in order to take account of initial property rights, guaranteeing that those who own a good initially will end up with a good that is either the same or better.

In the remainder of the paper, we prove several related results. In Section 4 we consider the problem of finding a minimum Pareto optimal matching, showing that this problem is NP-hard, though approximable within a factor of 2. In Section 5 we prove an interpolation result, showing that there exist Pareto optimal matchings of all sizes between a minimum and maximum Pareto optimal matching. Finally, in Section 6 we give a characterization, checkable in linear time, of instances that admit a unique Pareto optimal matching.

2 Preliminaries

We begin with a formal definition of the problem model under consideration. An instance I of the PARETO OPTIMAL MATCHING problem (POM) comprises a bipartite graph $G = (A, H, E)$, where $A = \{a_1, a_2, \dots, a_r\}$ is the set of agents and $H = \{h_1, h_2, \dots, h_s\}$ is the set of houses. For each $a_i \in A$, we denote by $A_i \subseteq H$ the vertices adjacent to a_i – these are referred to as the acceptable houses for a_i . Moreover a_i has a linear order over A_i . We let $n = r + s$ and $m = |E|$. Henceforth we assume that G contains no isolated vertices.

An assignment M is a subset of $A \times H$ such that $(a_i, h_j) \in M$ only if a_i finds h_j acceptable (i.e. $h_j \in A_i$). If $(a_i, h_j) \in M$, we say that a_i and h_j are matched to one another. For each $q \in A \cup H$, let $M(q)$ denote the assignees of q in M . An assignment M is a matching if $|M(q)| \leq 1$ for each $q \in A \cup H$. If $M(q) = \emptyset$, we say that q is unmatched in M , otherwise q is matched in M .

Let M be a matching in I . M is *unmatched* if there is no (agent,house) pair (a_i, h_j) such that a_i and h_j are both unmatched in M and $h_j \in A_i$. Also M is *unmatched* if there is no (agent,house) pair (a_i, h_j) such that a_i is matched in M , h_j is unmatched in M , and a_i prefers h_j to $M(a_i)$. Finally M is *unmatched* if M admits no *coalition*, which is a sequence of matched agents $C = \langle a_0, a_1, \dots, a_{k-1} \rangle$, for some $k \geq 2$, such that a_i prefers $M(a_{i+1})$ to $M(a_i)$ ($0 \leq i \leq k-1$) (here, and in the remainder of this paper, all subscripts are taken modulo k when reasoning about coalitions). The matching

$$M' = (M \setminus \{(a_i, M(a_i)) : 0 \leq i \leq k - 1\}) \cup \{(a_i, M(a_{i+1})) : 0 \leq i \leq k - 1\}$$

is defined to be the matching obtained from M by satisfying C .

The preferences of an agent extend to matchings as follows. Given two matchings M and M' , we say that an agent a_i prefers M' to M if either (i) a_i is matched in M' and unmatched in M , or (ii) a_i is matched in both M and M' and prefers $M'(a_i)$ to $M(a_i)$. Given this definition, we may define a relation \prec on the set of all matchings as follows: $M' \prec M$ if and only if no agent prefers M to M' , and some agent prefers M' to M . It is straightforward to then prove the following.

Proposition 1. *A matching M is Pareto optimal if and only if it is \prec -minimal.*

A matching is defined to be Pareto optimal if and only if it is \prec -minimal. Intuitively a matching is Pareto optimal if no agent a_i can be better off without requiring another agent a_j to be worse off. The following proposition gives necessary and sufficient conditions for a matching to be Pareto optimal.

Proposition 2. *A matching M is Pareto optimal if and only if M is maximal, trade-in-free and coalition-free.*

Let M be a Pareto optimal matching. If M is not maximal, then there exists an agent a_i and a house h_j , both unmatched in M , such that $h_j \in A_i$. Let $M' = M \cup \{(a_i, h_j)\}$. If M is not trade-in-free, then there exist an agent a_i and a house h_j , such that a_i is matched in M , h_j is unmatched in M , and a_i prefers h_j to $M(a_i)$. Let $M' = (M \setminus \{(a_i, M(a_i))\}) \cup \{(a_i, h_j)\}$. Finally if M admits some coalition C , let M' be the matching obtained by satisfying C . In all three cases, $M' \prec M$, a contradiction.

Conversely let M be a matching that is maximal, trade-in-free and coalition-free, and suppose for a contradiction that M is not Pareto optimal. Then there exists some matching M' such that $M' \prec M$. Let a_0 be any agent matched in M who prefers M' to M . Note that such an agent must exist, since M is maximal and at least one agent prefers M' to M .

It follows that $M'(a_0)$ is matched in M , say to a_1 , for otherwise M is not trade-in-free. Therefore, $M'(a_1) \neq M(a_1)$, and so a_1 must also prefer M' to M . Using this same argument, $M'(a_1)$ is matched in M , say to a_2 . We can continue in this manner finding a sequence of agents $\langle a_0, a_1, a_2, \dots \rangle$, where a_i prefers $M(a_{i+1})$ to $M(a_i)$. Since the number of agents is finite, this sequence must cycle, thereby contradicting the assumption that M is coalition-free. \square

Henceforth we will establish the Pareto optimality of a given matching by showing that the conditions of the above proposition are satisfied. For a given matching M , we can trivially check whether M satisfies the maximality and trade-in-free properties in $O(m)$ time. To check for the absence of coalitions, we construct the preference graph of M . This is a directed graph, denoted by $G(M)$, consisting of one vertex for each agent, with an edge from a_i to a_j whenever a_j is

matched in M and either (i) a_i is unmatched in M and finds $M(a_j)$ acceptable, or (ii) a_i is matched in M and prefers $M(a_j)$ to $M(a_i)$. It is clear that M is coalition-free if and only if $G(M)$ is acyclic. So we can perform this last check in $O(m)$ time by using a cycle-detection algorithm on $G(M)$. Putting these observations together, we have the following result.

Proposition 3. *Given an instance I of POM, we can find a maximum Pareto optimal matching M in $O(m)$ time.*

It is easy to construct an instance of POM in which the Pareto optimal matchings are of different sizes. For example let $A = \{a_1, a_2\}$ and let $H = \{h_1, h_2\}$. Suppose that a_1 prefers h_1 to h_2 , whilst a_2 finds only h_1 acceptable. Then both $M_1 = \{(a_1, h_1)\}$ and $M_2 = \{(a_1, h_2), (a_2, h_1)\}$ are Pareto optimal. Given this observation it is natural to consider the complexity of each of the problems of finding a maximum and minimum Pareto optimal matching. (Note that Greedy-POM produces M_1 given the agent ordering $\langle a_1, a_2 \rangle$, and produces M_2 given the agent ordering $\langle a_2, a_1 \rangle$.)

3 Maximum Pareto Optimal Matchings

In this section, we describe a three-phase algorithm for finding a maximum Pareto optimal matching, mirroring the three necessary and sufficient conditions in Proposition 2. We let I be an instance of POM, and we assume the notation and terminology introduced in Section 2. Phase 1 involves using the Hopcroft-Karp algorithm [7] to compute a maximum matching M in G . This phase, which guarantees that M is maximal, takes $O(\sqrt{nm})$ time and dominates the runtime. The final two phases transform M into a trade-in-free and coalition-free matching respectively. We describe these phases in more detail below.

3.1 Phase 2 of the Algorithm

In this phase, we transform M into a trade-in-free matching by repeatedly identifying and promoting agents that prefer an unmatched house to their existing assignment. Each promotion breaks the existing assignment, thereby freeing a house, which itself may be a preferred assignment for a different agent. With the aid of suitable data structures, we can ensure that the next agent and house can be identified efficiently.

For each house h , we maintain a linked list L_h of pairs (a, r) , where a is a matched agent who finds h acceptable, and r is the rank of h in a 's preference list. Initially the pairs in L_h involve only those matched agents a who prefer h to $M(a)$, though subsequently the pairs in L_h may contain agents a who prefer $M(a)$ to h . The initialization of these lists can be carried out using one traversal of the agent preference lists, which we assume are represented as doubly linked lists or arrays, in $O(m)$ time.

For each matched agent a , we also use this traversal to initialize a variable, denoted by $curr_a$, which stores the rank of $M(a)$ in a 's preference list. This

variable is maintained during the execution of the algorithm. We also assume that, for each matched agent a we store $M(a)$. One final initialization remains: construct a stack S of all unmatched houses h where L_h is non-empty. We now enter the loop described in Figure 1.

```

while  $S \neq \emptyset$ 
   $h := S.pop()$ ;
   $(a, r) := L_h.removeHead()$ ;
  if  $r < curr_a$ 
    //  $h$  is unmatched in  $M$ ,  $a$  is matched in  $M$  and prefers  $h$  to  $M(a)$ 
     $h' := M(a)$ ;
     $M := (M \setminus \{(a, h')\}) \cup \{(a, h)\}$ ;
     $curr_a := r$ ;
     $h := h'$ ;
  if  $L_h \neq \emptyset$ 
     $S.push(h)$ ;

```

Fig. 1. Phase 2 loop

During each loop iteration we pop an unmatched house h from S and remove the first pair (a, r) from the list L_h (which must be non-empty). If a prefers h to $M(a)$ (i.e. $r < curr_a$) then a is promoted from $h' = M(a)$ to h , also M and $curr_a$ are updated, and finally h' , which is now unmatched, is pushed onto S if $L_{h'}$ is non-empty. Otherwise h is pushed back onto S if L_h is non-empty.

Each iteration of the loop removes a pair from a list L_h . Since agent preference lists are finite and no new pair is added to a list L_h during a loop iteration, the while loop must eventually terminate with S empty. At this point no matched agent a would trade $M(a)$ for an unmatched house, and so M is trade-in-free. Additionally, M remains a maximum matching, since any agent matched at the end of Phase 1 is also matched at the end of Phase 2. Finally, it is clear that this phase runs in $O(m)$ time given the data structures described above.

3.2 Phase 3 of the Algorithm

In this phase, we transform M into a coalition-free matching. Recall that coalitions in M correspond to cycles in the envy graph $G(M)$. So a natural algorithm involves repeatedly finding and satisfying coalitions in $G(M)$ until no more coalitions remain. This algorithm has a runtime of $O(m^2)$, since there are $O(m)$ coalitions, and cycle-detection takes $O(m)$ time.

A better starting point for an efficient algorithm is Gale's Top Trading Cycles Method [14]. This method is also based on repeatedly finding and satisfying coalitions, however the number of iterations is reduced by the following observation: no agent assigned to his/her first choice can be in a coalition. We remove such agents from consideration, and since the houses assigned to them are no longer exchangeable, they can be deleted from the preference lists of the remaining agents. This observation can now be recursively applied to the reduced


```

for each matched agent  $a$  such that  $p(a) \neq M(a)$ 
   $P := \{a\};$  //  $P$  is a stack of agents
   $c(a) := 1;$  // counters record the number of times an agent is in  $P$ 
  while  $P \neq \emptyset$ 
     $a' := P.\text{pop}();$ 
     $p(a') :=$  most-preferred unlabelled house on preference list of  $a'$ ;
    if  $c(a') = 2$ 
       $C :=$  coalition in  $P$  containing  $a'$ ;
      satisfy  $C$ ;
      for each  $a'' \in C$ 
        label  $M(a'')$ ;
         $c(a'') := 0;$ 
         $P.\text{pop}();$ 
    else if  $p(a') = M(a')$ 
      label  $M(a')$ ;
       $c(a') := 0;$ 
    else
       $P.\text{push}(a');$ 
       $a'' := M(p(a'));$ 
       $c(a'') := c(a'') + 1;$ 
       $P.\text{push}(a'');$ 

```

Fig. 2. Phase 3 loop

preference lists. At some point, either no agents remain, in which case the matching is coalition-free, or no agent is assigned to his/her reduced first choice (i.e. the first choice on his/her reduced preference list).

In this last case, it turns out that there must be a coalition C in M , which can be found in $O(r)$ time by searching the envy graph restricted to reduced first-choice edges. After satisfying C , each agent in C is assigned to his/her reduced first choice. Therefore, no agent is in more than one coalition, giving $O(r)$ coalitions overall. The runtime of this preliminary implementation then is $\Omega(m + r^2)$. We now present a linear-time extension of Yuan's description of the Top Trading Cycles Method [15].

In our implementation, deletions of houses from agents' preference lists are not explicitly carried out. Instead, a house that is no longer exchangeable is labelled (all houses are initially unlabelled). For each agent a we maintain a pointer $p(a)$ to the first unlabelled house on a 's preference list – this is equivalent to the first house on a 's reduced preference list. Initially $p(a)$ points to the first house on a 's preference list, and subsequently $p(a)$ traverses left to right. Also, in order to identify coalitions, we initialize a counter $c(a)$ to 0 for each agent a . Then, we enter the main body of the algorithm, as given in Figure 2.

This algorithm repeatedly searches for coalitions, building a path P of agents (represented by a stack) in the (simulated) envy graph restricted to reduced first-choice edges. At each iteration of the while loop, we pop an agent a' from the stack and move up $p(a')$ if necessary. If P cycles (i.e. we find that $c(a') = 2$), there is a coalition C – the agents involved in C are removed from consideration

and the houses assigned to these agents are labelled (in practice the agents in C can be identified and C can be satisfied during the stack popping operations). Alternatively, if P reaches a dead-end (a' is already assigned to his/her first choice), this agent is removed from consideration and his/her assigned house is labelled. Otherwise, we keep extending the path by following the reduced first-choice edges.

At the termination of this phase we note that M is coalition-free by the correctness of the Top Trading Cycles Method [14]. Also M remains a maximum trade-in-free matching, since each agent and house matched at the end of Phase 2 is also matched at the end of Phase 3. Finally, it is clear this phase runs in $O(m)$ time given the data structures described above. We summarize the preceding discussion in the following theorem.

Theorem 1. *..... $O(\sqrt{nm})$ *

We now show that any improvement to the complexity of the above algorithm would imply an improved algorithm for finding a maximum matching in a bipartite graph. Without loss of generality, let $G = (A, H, E)$ be an arbitrary bipartite graph with no isolated vertices. Construct an instance I of POM with bipartite graph G , where each agent a 's preference list in I is an arbitrary permutation over a 's neighbours in G . By Theorem 1, any maximum Pareto optimal matching in I is also a maximum matching in G . Since I may be constructed from G in $O(m)$ time, the complexity of finding a maximum matching in a bipartite graph is bounded above by the complexity of finding a maximum Pareto optimal matching.

3.3 Initial Property Rights

Suppose that a subset A' of the agents already own a house. We now describe an modification of our algorithm, which ensures that every agent in A' ends up with the same house or better.

We begin with a matching M that pre-assigns every agent $a \in A'$ to his/her existing house h . We then truncate the preference list of each such a by removing all houses less preferable than $M(a)$. Now, we enter Phase 1, where we use the Hopcroft-Karp algorithm to exhaustively augment M into some matching M' . Members of A' must still be matched in M' , and since their preference lists were truncated, their new assignments must be at least as preferable as those in M . Note that M' may not be a maximum matching of A to H , however M' does have maximum cardinality among all matchings that respect the initial property rights. The remaining two phases do not move any agent from being matched to unmatched, and so the result follows immediately.

In the special case that all agents own a house initially (i.e. I is an instance of a housing market), it is clear that Phases 1 and 2 of the algorithm are not required. Moreover it is known that Phase 3 produces the unique matching that belongs to the core of the market [12], a stronger notion than Pareto optimality.

4 Minimum Pareto Optimal Matchings

In this section, we consider the problem of finding a minimum Pareto optimal matching. Let MIN-POM denote the problem deciding, given an instance I of POM and an integer K , whether I admits a Pareto optimal matching of size at most K . We firstly prove that MIN-POM is NP-complete via a reduction from MMM, which is the problem of deciding, given a graph G and an integer K , whether G admits a maximal matching of size at most K .

Theorem 2.

By Proposition 3, MIN-POM belongs to NP. To show NP-hardness, we give a reduction from the NP-complete restriction of MMM to subdivision graphs [6] (given a graph G , the subdivision graph of G is obtained by subdividing each edge $e = \{u, w\}$ into two edges $\{u, v_e\}, \{v_e, w\}$, where v_e is a new vertex corresponding to e).

Let $G = (V, E)$ (a subdivision graph) and K (a positive integer) be given as an instance of MMM. Then V is a disjoint union of two sets U and W , where each edge $e \in E$ joins a vertex in U to a vertex in W . Assume that $U = \{u_1, u_2, \dots, u_r\}$ and $W = \{w_1, w_2, \dots, w_s\}$. Without loss of generality assume that each vertex $u_i \in U$ has degree 2, and moreover assume that p_i and q_i are two sequences such that $p_i < q_i$, $\{u_i, w_{p_i}\} \in E$ and $\{u_i, w_{q_i}\} \in E$.

We create an instance I of MIN-POM as follows. Let A be the set of agents and let H be the set of houses, where $A = A^1 \cup A^2$, $A^t = \{a_1^t, a_2^t, \dots, a_r^t\}$ ($t = 1, 2$), $H = W \cup X$ and $X = \{x_1, x_2, \dots, x_r\}$. For each i ($1 \leq i \leq r$), we create preference lists for agents a_i^1 and a_i^2 as follows:

$$a_i^1 : x_i \ w_{p_i} \ w_{q_i} \qquad a_i^2 : x_i \ w_{q_i} \ w_{p_i}$$

We claim that G has a maximal matching of size at most K if and only if I has a Pareto optimal matching of size at most $K + r$.

For, suppose that M is a maximal matching in G of size at most K . We construct a set M' as follows. For any $u_i \in U$ that is unmatched in M , add the pair (a_i^1, x_i) to M' . Now suppose that $(u_i, w_j) \in M$. If $j = p_i$, add the pairs (a_i^1, w_j) and (a_i^2, x_i) to M' . If $j = q_i$, add the pairs (a_i^1, x_i) and (a_i^2, w_j) to M' . Clearly M' is a matching in I , and $|M'| = |M| + r \leq K + r$. It is straightforward to verify that, by the maximality of M in G , M' is Pareto optimal in I .

Conversely suppose that M' is a Pareto optimal matching in I of size at most $K + r$. For each i ($1 \leq i \leq r$), either $(a_i^1, x_i) \in M'$ or $(a_i^2, x_i) \in M'$, for otherwise M' is not trade-in-free. Hence we may construct a matching M in G as follows. For each i ($1 \leq i \leq r$), if $(a_i^t, w_j) \in M'$ for some t ($1 \leq t \leq 2$), add (u_i, w_j) to M . Then $|M| = |M'| - r \leq K$. The maximality of M' clearly implies that M is maximal in G . \square

For a given instance I of POM with bipartite graph G , we denote by $p^-(I)$ and $p^+(I)$ the sizes of a minimum and maximum Pareto optimal matching in I respectively. Similarly, we denote by $\beta_1^-(G)$ and $\beta_1(G)$ the sizes of a minimum maximal and a maximum matching in G respectively. It is known that

$\beta_1^-(G) \geq \beta_1(G)/2$ [9]. By Proposition 2, Pareto optimal matchings in I are maximal matchings in G . Hence, by Theorem 1, we have that $\beta_1^-(G) \leq p^-(I) \leq p^+(I) = \beta_1(G)$. It is therefore immediate that, for a given instance I of POM, the problem of finding a minimum Pareto optimal matching is approximable within a factor of 2.

5 Interpolation of Pareto Optimal Matchings

In this section, we prove that, for a given instance I of POM, there are Pareto optimal matchings of all sizes between $p^-(I)$ and $p^+(I)$.

Given a matching M , an augmenting path P for M is an alternating sequence of distinct agents and houses $\langle a_1, h_1, a_2, \dots, a_k, h_k \rangle$, where a_1 and h_k are unmatched in M , $h_i \in A_i$, and $M(a_{i+1}) = h_i$ ($1 \leq i \leq k - 1$). We associate with each such augmenting path a vector $\langle \dots \rangle_P$, whose i th component contains the rank of a_i for h_i . Given two augmenting paths P and Q for M , we say that $P \triangleleft Q$ if (i) both P and Q begin from the same agent, and (ii) $\langle \dots \rangle_P$ is lexicographically less than $\langle \dots \rangle_Q$. Also for paths P and Q , we define three operations: $\langle \dots \rangle_P(v)$ is the substring of P from a_1 to $v \in P$, $\langle \dots \rangle_P(v)$ is the substring of P from $v \in P$ to h_k , and $P \cdot Q$ denotes the concatenation of P and Q .

Theorem 3. *For any instance I of POM, for any integer k , $p^-(I) \leq k \leq p^+(I)$ implies the existence of a Pareto optimal matching of size k .*

Let M be any Pareto optimal matching such that $|M| < p^+(I)$, and let M' be the matching that results from augmenting M along some \triangleleft -minimal augmenting path P . We will show in turn that M' is maximal, trade-in-free and coalition-free; the result then follows by induction.

If M' is not maximal, then clearly we contradict the maximality of M . Now suppose that M' is not trade-in-free. Then there exists an agent a and house h , where a is matched in M' , h is unmatched in M' , and a prefers h to $M'(a)$. Since h is also unmatched in M , a must be in P , for otherwise $M(a) = M'(a)$, and M is not trade-in-free. But then $P' = \langle \dots \rangle_P(a) \cdot \langle h \rangle$ is an augmenting path for M , contradicting the \triangleleft -minimality of P .

Finally suppose for a contradiction that M' is not coalition-free. Then there exists a coalition $C = \langle a_0, a_1, \dots, a_{k-1} \rangle$ with respect to M' . At least one agent in P must also be in C , for otherwise M is not coalition-free. Let a_i be the first such agent in P . We establish some properties of $M'(a_{i+1})$. Firstly, $M'(a_{i+1})$ must be matched in M , for otherwise M admits the augmenting path $\langle \dots \rangle_P(a_i) \cdot \langle M'(a_{i+1}) \rangle$, contradicting the \triangleleft -minimality of P . Also, $M'(a_{i+1})$ cannot appear before a_i in P , for otherwise a_i is not the first agent in P to be in C . Lastly, $M'(a_{i+1})$ cannot appear after a_i in P , for otherwise M admits the augmenting path $\langle \dots \rangle_P(a_i) \cdot \langle \dots \rangle_P(M'(a_{i+1}))$, contradicting the \triangleleft -minimality of P . So, it must be the case that $M'(a_{i+1})$ is matched in M and does not appear in P . Let a_{i+j} be the first agent in C after a_i , such that a_{i+j} is in P . Note that $a_{i+j} \neq a_{i+1}$ by the above properties of $M'(a_{i+1})$, but since C is a cycle, $a_{i+j} = a_i$ is possible.

It follows that the subsequence $S = \langle M'(a_{i+1}), a_{i+1}, \dots, M'(a_{i+j-1}), a_{i+j-1} \rangle$ of C is disjoint from P , and so $P' = \dots P(a_i) \cdot S \cdot \dots P(M'(a_{i+j}))$ is a valid augmenting path of M . But then P' contradicts the \leftarrow -minimality of P , since a_i prefers $M'(a_{i+1})$ to $M'(a_i)$. \square

Corollary 1. *Let $I = (A, H, E)$ be an instance of the house allocation problem with $|A| = k$ and $|H| = k + 1$. Let M be a Pareto optimal matching in I . Then M can be found in $O(m)$ time.*

Let G be the bipartite graph in I , with edges in M directed from H to A , and edges not in M directed from A to H . Also associate with each non-matching edge (a_i, h_j) the rank of a_i for h_j . We search for a \leftarrow -minimal augmenting path by performing an ordered depth first search of G starting from the set of unmatched agents, where for each agent a in the search, we explore outgoing edges from a in increasing order of rank. In general, ordered depth-first search is asymptotically slower than depth-first search. However, the $O(m)$ result holds, since each preference list is already given in increasing order of rank. \square

We remark that the results of this section extend to the case where a subset of the agents have initial property rights.

6 Uniqueness of Pareto Optimal Matchings

In this section, we give a characterization of instances with no initial property rights that admit a unique Pareto optimal matching. This is based on the concept of a signature of a Pareto optimal matching.

If a matching M is Pareto optimal, the envy graph $G(M)$ contains no cycles, and therefore admits a topological ordering. We say that a reversed topological ordering of $G(M)$, denoted by $\sigma(M)$, is a signature of M . The next lemma will help us establish that the signature of a matching is unique for that matching. This lemma is similar to [1–Lemma 1], though the proof here, which uses the concept of a signature, is much simpler.

Lemma 1. *Let $I = (A, H, E)$ be an instance of the house allocation problem. Let M be a Pareto optimal matching in I . Then M is the unique Pareto optimal matching in I with signature $\sigma(M)$.*

Let M be an arbitrary Pareto optimal matching in I . We claim that by processing the agents in order of $\sigma(M)$, the greedy algorithm returns M .

Suppose for a contradiction that Greedy-POM returns a matching $M' \neq M$. It follows that since M' is Pareto optimal, some agent must prefer M' to M . Let a be the first such agent in $\sigma(M)$.

Now, $M'(a)$ must be matched in M , say to a' , for otherwise M is not maximal (if a is unmatched in M), or M is not trade-in-free (if a is matched in M). $G(M)$ must therefore contain an edge from a to a' , meaning that a' precedes a in $\sigma(M)$. At the time a' is processed by Greedy-POM, $M'(a)$ is unmatched (since it is assigned later to a). So, a' must prefer $M'(a')$ to $M(a') = M'(a)$, contradicting the assumption that a was the first such agent in $\sigma(M)$. \square

Corollary 2. *Let I be an instance of the house allocation problem. Then I admits a unique Pareto optimal matching if and only if I admits a matching in which every agent is matched to his/her first choice.*

We can now present a necessary and sufficient condition, checkable in linear time, for a POM instance to admit a unique Pareto optimal matching.

Theorem 4. *Let I be an instance of the house allocation problem. Then I admits a unique Pareto optimal matching if and only if I admits a matching M in which every agent is matched to his/her first choice.*

Let M be the unique Pareto optimal matching in I . Since every agent permutation is a signature of M , $G(M)$ contains no edges. Then every agent must be matched to his/her first choice.

Conversely, let M be a matching in I in which every agent is matched with his/her first choice. Then if M' is any matching in I such that $M' \neq M$, it follows that $M \prec M'$. Hence M is the unique Pareto optimal matching in I . \square

7 Concluding Remarks

We conclude with an open problem. The basic POM definition given in Section 2 can be generalized by permitting agents to contain \dots in their preference lists (i.e. to rank equally two or more houses). In this context the definition of the relation \prec is the same as that given in Section 2, and hence the definition of Pareto optimality remains unchanged. A maximum Pareto optimal matching can be found in $O(\sqrt{nm} \log n)$ time using a similar reduction to the Assignment problem as described in Section 1 (in this case $rank_{a,h}$ is the number of houses that a prefers to h). However is the problem of finding a maximum Pareto optimal matching solvable in $O(\sqrt{nm})$ time?

References

1. A. Abdulkadiroğlu and T. Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.
2. A. Abdulkadiroğlu and T. Sönmez. House allocation with existing tenants. *Journal of Economic Theory*, 88:233–260, 1999.
3. X. Deng, C. Papadimitriou, and S. Safra. On the complexity of equilibria. *Journal of Computer and System Sciences*, 67(2):311–324, 2003.
4. S.P. Fekete, M. Skutella, and G.J. Woeginger. The complexity of economic equilibria for house allocation markets. *Inf. Proc. Lett.*, 88:219–223, 2003.
5. H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.
6. J.D. Horton and K. Kilakos. Minimum edge dominating sets. *SIAM Journal on Discrete Mathematics*, 6:375–387, 1993.
7. J.E. Hopcroft and R.M. Karp. A $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

8. A. Hylland and R. Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.
9. B. Korte and D. Hausmann. An analysis of the greedy heuristic for independence systems. *Annals of Discrete Mathematics*, 2:65–74, 1978.
10. R.W. Irving, T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. Rank-maximal matchings. *Proceedings of SODA '04*, pages 68–75. ACM-SIAM, 2004.
11. A.E. Roth. Incentive compatibility in a market with indivisible goods. *Economics Letters*, 9:127–132, 1982.
12. A.E. Roth and A. Postlewaite. Weak versus strong domination in a market with indivisible goods. *Journal of Mathematical Economics*, 4:131–137, 1977.
13. A.E. Roth and M.A.O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis*. Cambridge University Press, 1990.
14. L. Shapley and H. Scarf. On cores and indivisibility. *Journal of Mathematical Economics*, 1:23–37, 1974.
15. Y. Yuan. Residence exchange wanted: a stable residence exchange problem. *European Journal of Operational Research*, 90:536–546, 1996.
16. L. Zhou. On a conjecture by Gale about one-sided matching problems. *Journal of Economic Theory*, 52(1):123–135, 1990.

Property-Preserving Data Reconstruction^{*}

Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu

Dept. Comp. Sci., Princeton University, Princeton NJ 08544, USA
{nailon, chazelle, csesha, dingliu}@cs.princeton.edu

Abstract. We initiate a new line of investigation into *online property-preserving data reconstruction*. Consider a dataset which is assumed to satisfy various (known) structural properties; eg, it may consist of sorted numbers, or points on a manifold, or vectors in a polyhedral cone, or codewords from an error-correcting code. Because of noise and errors, however, an (unknown) fraction of the data is deemed *unsound*, ie, in violation with the expected structural properties. Can one still query into the dataset in an online fashion and be provided data that is *always* sound? In other words, can one design a filter which, when given a query to any item I in the dataset, returns a *sound* item J that, although not necessarily in the dataset, differs from I as infrequently as possible. No preprocessing should be allowed and queries should be answered online. We consider the case of a monotone function. Specifically, the dataset encodes a function $f : \{1, \dots, n\} \mapsto \mathbf{R}$ that is at (unknown) distance ε from monotone, meaning that f can—and must—be modified at εn places to become monotone.

Our main result is a randomized filter that can answer any query in $O(\log^2 n \log \log n)$ time while modifying the function f at only $O(\varepsilon n)$ places. The amortized time over n function evaluations is $O(\log n)$. The filter works as stated with probability arbitrarily close to 1. We also provide an alternative filter with $O(\log n)$ worst case query time and $O(\varepsilon n \log n)$ function modifications.

1 Introduction

It is a fact of (computing) life that massive datasets often come laden with varying degrees of reliability. Errors might be inherent to the data acquisition itself (faulty sensors, white/bursty noise, aliasing), or to data processing (roundoff errors, numerical instability, coding bugs), or even to intrinsic uncertainty (think of surveys and poll data). Classical error correction postulates the existence of exact data and uses redundancy to provide recovery mechanisms in the presence of errors. Mesh generation in computer graphics, on the other hand, will often deal with reconstruction mostly on the basis of esthetic criteria, while signal processing might filter out noise by relying on frequency domain models.

^{*} This work was supported in part by NSF grants CCR-998817, 0306283, ARO Grant DAAH04-96-1-0181.

In the case of geometric datasets, reconstruction must sometimes seek to enforce structural properties. Early work on geometric robustness [5,9] pointed out the importance of topological consistency. For example, one might want to ensure that the output of an imprecise, error-prone computation of a Voronoi diagram is a Voronoi diagram (albeit that of a slightly perturbed set of points). Geometric algorithm design is notoriously sensitive to structure: Dimensionality, convexity, and monotonicity are features that often impact the design and complexity of geometric algorithms. Consider a computation that requires that the input be a set of points in convex position. If the input is noisy, convexity might be violated and the algorithm might crash. Is there a filter that can be inserted between the algorithm (the client) and the dataset so that: (i) the client is always provided with a point set in convex position; and (ii) the “filtered” data differs as little as possible from the original (noisy) data? In an offline setting, the filter can always go over the entire dataset, compute the “nearest” convex-position point set, and store it as its filtered dataset. This is unrealistic in the presence of massive input size, however, and only online solutions requiring no preprocessing at all can be considered viable. We call this

. Besides convexity, other properties we might wish to preserve include low dimensionality and angular constraints:

- Consider a dataset consisting of points on a low-dim manifold embedded in very high dimensional space. Obviously, the slightest noise is enough to make the point set full-dimensional. How to “pull back” points to the (unknown) manifold online can be highly nonobvious.
- Angle constraints are of paramount importance in industrial/architectural design. Opposite walls of a building have a habit of being parallel, and no amount of noise and error should violate that property. Again, the design of a suitable filter to enforce such angular constraints online is an interesting open problem.

In this paper we consider one of the simplest possible instances of online property-preserving reconstruction: monotone functions. Sorted lists of numbers are a requirement for all sorts of operations. A binary search, for example, will easily err if the list is not perfectly sorted. In this case of property-preserving data reconstruction, the filter must be able to return a value that is consistent with a sorted list and differs from the original as little as possible. (An immediate application of such a filter is to provide robustness for binary searching in near-sorted lists.)

We formalize the problem. Let $f : \{1, \dots, n\} \mapsto \mathbf{R}$ be a function at an unknown distance ε from monotonicity, which means that f can (and must) be modified at εn places to become monotone. Figure 1 illustrates the filter in action. To avoid confusion, we use the term “query” to denote interaction between the client and the filter, and “lookup” to denote interaction between the filter and the dataset. Given a query x , the filter generates lookups a, b, c, \dots to the dataset, from which it receives the values $f(a), f(b), f(c), \dots$, and then computes a value $g(x)$ such that the function g is monotone and differs from f in at most $c\varepsilon n$ places, for some c (typically constant, but not necessarily so). We note two things.

1. Once the filter returns $g(x)$ for some query x , it commits to this value and must return the same value upon future queries.
2. The filter may choose to follow a multi-round protocol and adaptively generate lookups to the dataset depending on previous results. The function $g(x)$ is defined on the fly, and it can depend on both the queries and on random bits. Therefore, after the first few queries, g might only be defined on a small fraction of the domain. At any point in time, if k distinct x_i 's have been queried so far, then querying the remaining x_i 's (whether the client does it or not) while honoring past commitments leads to a monotone function close enough to f .

It is natural to measure the performance of the filter with respect to two functions. A $(p(n, \varepsilon), q(n))$ -filter performs $O(p(n, \varepsilon))$ lookups per query, and returns a function g that is at a distance of at most $q(n)\varepsilon$ from monotonicity, with high probability. The lookup-per-query guarantee can be either amortized or in worst case (the running times are deterministic). Ideally, we would like $p(n, \varepsilon)$ to depend only on n , and $q(n)$ to be constant. There is a natural tradeoff between p and q : we expect q to decrease as p increases. We will see an example of this in this work.

Theorem 1. *For any $\delta > 0$, there exists a filter with $(\log^2 n \log \log n, 2 + \delta)$ lookups per query and a distance guarantee of $O(\log n)$.*

We also provide an alternative filter with a better lookups-per-query guarantee and a worse distance guarantee.

Theorem 2. *For any $\delta > 0$, there exists a filter with $(\log n, O(\log n))$ lookups per query and a distance guarantee of $O(\log n)$.*

It is important to note that, in this work, we think of the client as an adversary. That is, the filter's guarantees must hold for all sequences of client queries. However, in some cases it might be useful to assume the client's queries are drawn from some known probability distribution. We will see that the filter can take advantage of this.

Theorem 3. *For any $\delta > 0$, there exists a filter with $(1, O(\log n))$ lookups per query and a distance guarantee of $O(\log n)$.*

We are not aware of any previous work on this specific problem. Of course, property testing is by now a well studied area [4, 8], with many nontrivial results regarding combinatorial, algebraic, and geometric problems [2, 3, 7]. More recent work [1, 6] has provided sublinear algorithms for estimating the distance of a function to monotone. We use ideas from [1] in this work.

2 The $(\log^2 n \log \log n, 2 + \delta)$ -Filter

We use the following notation in what follows. The distance between two functions f_1 and f_2 over the domain $\{1, \dots, n\}$ is defined as the fractional size of

domain points on which they disagree. The function f and the domain size n which are the input to the problem (the dataset in Figure 1) are fixed. We use ε to denote the distance of f from monotonicity, and \hat{f} to denote the monotone function closest to f . So the distance between f and \hat{f} is ε , and \hat{f} minimizes the distance between f and any monotone function. We use g to denote the function outputted by the filter.

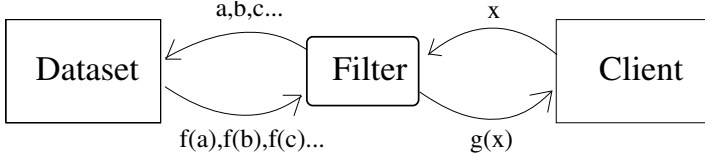


Fig. 1. The property-preserving reconstruction filter: g is sound and differs from f in few places

2.1 Preliminaries

Proving Theorem 1 requires a few preliminaries, beginning with these definitions:

- δ -bad: Given $0 < \delta < 1/2$, the integer i is called δ -bad if there exists $j > i$ such that

$$\left| \left\{ i \leq k \leq j \mid f(k) < f(i) \right\} \right| \geq (1/2 - \delta)(j - i + 1)$$

or, similarly, $j < i$ such that

$$\left| \left\{ j \leq k \leq i \mid f(k) > f(i) \right\} \right| \geq (1/2 - \delta)(i - j + 1) .$$

Otherwise the integer i is called δ -good.

- a -good: Let \mathcal{D} be the joint distribution of m independent 0/1 random variables x_1, \dots, x_m , which can be sampled independently. If $\mathbf{E}[x_i] \leq a$ for all i , then \mathcal{D} is called a -good; else it is a -bad.

Lemma 1. (Ailon et al. [1]) *If $a < b$, \mathcal{D} is a -good with probability $\geq 2/3$ and $O(m)$ samples.*

In the following we use the algorithm of Lemma 1 to test, with high probability of success, whether a given integer i is δ -bad or 2δ -good, for any fixed $\delta > 0$. Given an interval $[u, v]$, we define two 0/1 random variables $\alpha[u, v]$ and $\beta[u, v]$: given a random integer $j \in [u, v]$, $\alpha[u, v] = 1$ (resp. $\beta[u, v] = 1$) iff $f(u) > f(j)$ (resp. $f(j) > f(v)$). The algorithm `bad-good-test` (Fig. 2) tests if a given integer i is δ -bad or 2δ -good.

Lemma 2. *If $\delta > 0$, $\text{bad-good-test}(k, i, \delta)$ returns δ -good with probability $\geq 1 - 1/k$ and $O(\log n \log k)$ samples.*

If integer i is 2δ -good then the expectation of every $x_{2j-1}^{(i)}$ or $x_{2j}^{(i)}$ defined in `bad-good-test` is at most $1/2 - 2\delta$, and so the distribution \mathcal{D} is $(1/2 - 2\delta)$ -light. On the other hand, if i is δ -bad, then there exists some $x_{2j-1}^{(i)}$ or $x_{2j}^{(i)}$ with expectation at least $(1/2 - \delta)/(1 + \delta) \geq 1/2 - 3\delta/2$, and so \mathcal{D} is $(1/2 - 3\delta/2)$ -heavy. The algorithm from Lemma 1 distinguishes between $(1/2 - 2\delta)$ -light and $(1/2 - 3\delta/2)$ -heavy with probability $2/3$ in $O(\log n)$ time. Since we repeat it $c \log k$ times and take a majority vote, a standard Chernoff bound argument shows that `bad-good-test` fails with probability at most $1/k$. \square

```

bad-good-test ( $f, i, \delta, k$ )

repeat the following  $c \log k$  times for a big enough constant  $c$ 
  for each  $1 \leq j \leq (2/\delta) \ln n$ 
    define  $x_{2j-1}^{(i)} = \alpha[i, i + (1 + \delta)^j]$  and  $x_{2j}^{(i)} = \beta[i - (1 + \delta)^j, i]$ 
    let  $\mathcal{D}$  be the distribution  $(x_1, x_2, \dots)$ 
  if majority of above tests output  $(1/2 - 2\delta)$ -light
  then output  $2\delta$ -good
  else output  $\delta$ -bad
    
```

Fig. 2. Testing if an integer i is δ -bad or 2δ -good

Lemma 3. *Let f be a function on $[1, n]$ and \hat{f} be a function on $[1, n]$ such that $\|f - \hat{f}\|_{\infty} \leq (2 + O(\delta))\varepsilon n^{-\delta}$. (Ailon et al. [1])*

Finally, we let $\delta > 0$ denote an arbitrarily small positive real. Choosing a small enough δ will satisfy the distance guarantee of Theorem 1.

2.2 The Algorithm

We now describe the algorithm `monotonize`. Our goal, as described above, is: given a fixed $\delta > 0$, compute a function g online such that: (1) g is monotone; (2) g is $((2 + O(\delta))\varepsilon)$ -close to f . Specifically, on query i , `monotonize` computes $g(i)$ in time $O(\log^2 n \log \log n)$. Whenever `monotonize` outputs a value $g(i)$, this value must be recorded to ensure consistency. The procedure will therefore hold an internal data structure that will record past commitments. The data structure can be designed to allow efficient retrievals, but we omit the details because we are mainly interested in the number of f -lookups it performs, and not the cost of other operations.

Given a query i , `monotonize` first checks whether i was committed to in the past, and returns that commitment in that case. If not, more work should be done. In virtue of Lemma 3, `monotonize` tries to keep the f values at δ -good

integers and change the values for other queries. We will use `bad-good-test` to decide whether i is bad or good.

Suppose now we decide that i is δ -bad and hence $g(i)$ needs a value that might be different from $f(i)$. Ideally, we would like to find the closest δ -good integers l (to the left of i) and r (to the right of i) and assign $g(i)$ to some value between $f(l)$ and $f(r)$. Because of the sublinear time constraint, we slightly relax this condition. Instead, the idea is to find an interval I_0 around i such that the fraction of 2δ -good integers in I_0 is at least $\Omega(\delta)$, but their fraction in a slightly smaller interval is $O(\delta)$. This ensures that such an interval can be detected through random sampling and that there are not many 2δ -good integers between i and any 2δ -good integer in this interval (a relaxation of the closest condition).

We will search for a good interval within the interval determined by the closest committed values on the left and right of i . Denote this interval by $[l, r]$. Once such a good interval I_0 is found, we try to find a value x that is sandwiched between values of f evaluated at two δ -good points in I_0 . Finding x is done in `find-good-value` (Figure 3). We commit to the value x on g restricted to I_0 . If no good intervals are found, we spread the value of $g(l)$ on g in the interval $[l + 1, r - 1]$.

`find-good-value` (f, I, δ)

```

set  $L$  as an empty list
randomly select  $c\delta^{-1} \log n$  integers from  $I$  for a big enough constant  $c$ .
for each  $j$  in random sample
    if bad-good-test ( $f, j, \delta, \delta^{-2}$ ) returns “ $2\delta$ -good”
        then append  $f(j)$  to  $L$ 
return median value of  $L$ 
    
```

Fig. 3. Finding a good value in an interval

Lemma 4. $\mathbb{E}[\text{find-good-value}(f, I, \delta)] \leq \frac{1 - 1/n^3}{\delta} \max_{i_1, i_2 \in I} |f(i_1) - f(i_2)| + O(\log^2 n)$

The expected number X of δ -bad samples for which `bad-good-test` returns “ 2δ -good” is at most $c\delta(1 - \delta) \log n$, by Lemma 2. The expected total number Y of samples for which `bad-good-test` returns “ 2δ -good” is at least $c(1 - \delta^2) \log n$. The probability that X exceeds $Y/2$ is at most $1/n^3$ if c is chosen large enough, using Chernoff bounds. Therefore, with probability at least $1 - 1/n^3$, more than half the values that are appended to the list L are the function f evaluated at δ -good points (“good values”). By taking the median of

values in L , in such a case, we are guaranteed to get a value sandwiched between two good values. The time bound follows from Lemma 2. \square

```

monotonize ( $f, \delta, i$ )

  if  $g(i)$  was already committed to then return  $g(i)$ 
  if bad-good-test( $f, i, \delta, n^3$ ) returns ‘‘2 $\delta$ -good’’
    then commit to  $g(i) = f(i)$  and return  $g(i)$ 

  let  $l$  be closest committed index on left of  $i$  (0 if none)
  let  $r$  be closest committed index on right of  $i$  ( $n+1$  if none)
  (*)
  set  $j_{max} = \lfloor \ln(i-l)/\ln(1+\delta) \rfloor$  and  $j_{min} = 0$ .
  while  $j_{max} - j_{min} > 1$ 
    set  $j = \lfloor (j_{max} + j_{min})/2 \rfloor$ ,  $I = [i - (1+\delta)^j, i]$ 
    choose random sample of size  $c\delta^{-1} \log n$  from  $I$ , for large  $c$ 
    for each point  $i'$  in sample
      run bad-good-test( $f, i', \delta, c_1$ ), for large  $c_1$ 
    if number of ‘‘2 $\delta$ -good’’ outputs is  $\geq \frac{3}{2}c \log n$ 
      then set  $j_{max} = j$ 
    else set  $j_{min} = j$ 
  if  $j_{max} \neq \lfloor \ln(i-l)/\ln(1+\delta) \rfloor$ 
    then set  $I_l = [i - (1+\delta)^{j_{max}}, i]$  and  $val_l = \text{find-good-value}(f, I_l, \delta)$ 
    else set  $val_l = g(l)$  and  $I_l = [l+1, i]$ 
  (**)
  repeat lines (*)...(**) for right side of  $i$ , obtaining  $val_r$  and  $I_r$ 

  choose  $val_l < y < val_r$  and commit to  $y$  on  $I_l \cup \{i\} \cup I_r$ 
  return  $y$ 

```

Fig. 4. Computing a monotone function online

To find a good interval, we do a binary search among all the intervals of length $(1+\delta)^j$ ($j = 0, 1, \dots$) starting or ending at i , that is, $[i, i + (1+\delta)^j]$ and $[i - (1+\delta)^j, i]$. There are $O(\log n)$ such intervals, and thus the running time is $O(\log \log n)$ times the time spent for each interval. The overall algorithm **monotonize** is shown in Figure 4. The following claim together with a suitable rescaling of δ concludes the proof of the first part of Theorem 1.

Given any $0 < \delta < \frac{1}{2}$, with probability $1 - 1/n$, **monotonize** computes a monotone function g that is within distance $(2+\delta)\varepsilon$ to f . Given a query i , $g(i)$ is computed online in time $O(\log^2 n \log \log n)$, when δ is assumed to be fixed.

First we analyze the running time. The **bad-good-test** in line 3 takes $O(\log^2 n)$ time. If the algorithm determines that i is δ -bad, then the while-loops run $O(\log \log n)$ times. In one iteration of the while-loop, the algorithm

calls `bad-good-test` $O(\log n)$ times. Each call takes $O(\log n)$ time by Lemma 2. Therefore, the time complexity of the while-loop is $O(\log^2 n \log \log n)$. By Lemma 4, the running time of the call to `find-good-integer` is $O(\log^2 n)$. The time complexity of the algorithm is therefore $O(\log^2 n \log \log n)$.

Let us first look at the while-loop. If I has more than 2δ -fraction of 2δ -good integers, then the number of "2 δ -good" outputs is $< \frac{3}{2}c \log n$ with inverse polynomial probability. This can be shown through Chernoff bounds. On the other hand, if I has less than δ -fraction of 2δ -good integers, then the number of "2 δ -good" outputs is $> \frac{3}{2}c \log n$ with inverse polynomial probability. Therefore, we can assume that -

- When `find-good-value` is called on interval I_l , I_l has at least a δ -fraction of 2δ -good integers.
- The interval $I_{min} = [i - (1 + \delta)^{j_{min}}, i]$ has at most 2δ -fraction of 2δ -good integers.

Both these events hold with probability $1 - 1/n^c$, for some large constant c . These events occur totally at most a polynomial number of times. We can also assume that, during the execution of the algorithm, the first call of `bad-good-test` (in line 3) correctly distinguishes between δ -bad and 2δ -good.

As shown in Lemma 2, this holds with probability $1 - 1/n^3$. This is totally called at most n times. By a union-bound, all of the above events occur with probability $1 - 1/n^d$, for some positive constant d .

To show that the function g is monotone, we first note that if `bad-good-test` outputs "2 δ -good" for i (leading to $g(i)$ being set to $f(i)$), then i is not δ -bad. If i is bad, then val_l lies between the value at two δ -good points in I_l . If val_l is assigned to all the values of g in I_l , then g would be monotone with respect to all the values at the δ -good points already committed to. Similarly, val_r can be assigned to the values of g in I_r without disturbing monotonicity. Therefore, since the algorithm assigns some value between val_l and val_r to $I_l \cup \{i\} \cup I_r$, g is remains monotone.

Finally we show that g is within distance $(2+\delta)\epsilon$ to f . We earlier showed that for $I_{min} = [i - (1 + \delta)^{j_{min}}, i]$, the fraction of 2δ -good integers in I_{min} is at most 2δ . Since by the end of the algorithm $j_{max} \leq j_{min} + 1$, the fraction of 2δ -good integers in $I_r = [i, i + (1 + \delta)^{j_{max}}]$ (or $I_l = [i - (1 + \delta)^{j_{max}}, i]$) is at most 4δ . In other words, each time we make a total of $|I_l \cup \{i\} \cup I_r|$ corrections to f at least a $(1 - 4\delta)$ -fraction of these changes are made on 2δ -bad integers. By Lemma 2.4 in [1], the total number of 2δ -bad integers is at most $(2 + 10\delta)\epsilon n$. So the total number of changes we made on f is at most $(2 + 10\delta)\epsilon n / (1 - 4\delta) \leq (2 + c\delta)\epsilon n$ for some constant c . This concludes the proof. \square

2.3 Achieving Logarithmic Amortized Query Time

In this section we show how to modify the algorithm to achieve better amortized query time. The worst case query time for a single query remains the same. We need a technical lemma first.

Lemma 5. $\dots 1/2 > \delta > 0, \dots i \dots \delta \dots l, r \dots \delta$
 $\dots l < i < r \dots i'$
 $\dots [l, r]$

If $f(i) < f(l)$, then we claim that l is a witness to i 's badness. In fact, since $f(l)$ and $f(i)$ is a violating pair, it is immediate that at least one of them is 0-bad with respect to the interval $[l, i]$. Since l is δ -good, i must be 0-bad (and hence δ -bad) with respect to $[l, i]$. In this case, l is a witness to i 's badness. Similarly, r will be a witness if $f(i) > f(r)$. In the following we assume that $f(l) < f(i) < f(r)$.

Let w be a witness to i 's badness. Without loss of generality, assume that $w < i$. If $w \geq l$ then we are done, so let $w < l$. Since i is δ -bad and l is δ -good, we know that: number of violations in $[w, l]$ with respect to l is $< (1/2 - \delta)(l - w + 1)$; number of violations in $[w, i]$ with respect to i is $\geq (1/2 - \delta)(i - w + 1)$. We also know that each violation in $[w, l]$ with respect to i is also a violation with respect to l , so the number of violations with respect to i in $[l + 1, i]$ is $> (1/2 - \delta)(i - l) = (1/2 - \delta)(i - (l + 1) + 1)$. This shows that i has a witness to its badness in $[l + 1, i]$. \square

The improvement on amortized query time comes from the following strategy: each time the algorithm answers a client query, it also generates a new query by itself and answers that query. This self query is completely independent of all the client queries, and we call it an *oblivious* query.

The oblivious queries are generated based on the balanced binary tree on $[1, n]$. The root of this tree is $\lfloor n/2 \rfloor$. The left subtree of the root corresponds to the interval $[1, \lfloor n/2 \rfloor - 1]$, and similarly the right subtree corresponds to $[\lfloor n/2 \rfloor + 1, n]$. The two subtrees are then defined recursively. This tree is denoted by T .

The oblivious queries are generated according to the following order. We start from the root of T and scan its elements one by one by going down level by level. Within each level we scan from left to right. This defines an ordering of all integers in $[1, n]$ which is the order to make oblivious queries. This ordering ensures that, after the $(2^k - 1)$ -th oblivious query, $[1, n]$ is divided by all the oblivious queries into a set of disjoint intervals of length at most $n/2^k$. Each oblivious query is either a δ -good integer itself in which case `monotonize` returns at line 6, or it causes two δ -good integers being outputted (val_l and val_r in `monotonize`). These two δ -good integers lie on the left and right side of the oblivious query, respectively. This shows that after the $(2^k - 1)$ -th oblivious query, $[1, n]$ is divided by some δ -good integers into a set of smaller intervals each of length at most $n/2^k$.

Based on Lemma 5, whenever we call `bad-good-test` (in `find-good-integer` or `monotonize`) to test the badness of an integer i , we only need to search for a witness within a smaller interval $[l, r]$ such that l (resp. r) is the closest δ -good integer on the left (resp. right) of i . As explained above, these δ -good integers come as by-products of oblivious queries. This will reduce the running time of `bad-good-test` to $O(\log n_i \log k)$ (to achieve success probability at least $1 - 1/k$), where $n_i = r - l + 1$. Accordingly, the time spent on binary searching intervals in `monotonize` is reduced to $O(\log \log n_i)$. By the distribution of oblivious queries, for the j -th client query where $2^{k-1} \leq j < 2^k$, the running time of `monotonize` is now $O(\log n \log \frac{n}{2^k} \log \log \frac{n}{2^k})$. The same is true for the j -th oblivious query.

To bound the amortized running time, it suffices to focus on the smallest m such that all n distinct queries appear in the first m queries (including both client and oblivious queries). We can also ignore repetition queries (those that

have appeared before) since each one only takes $O(\log n)$ time using standard data structure techniques. Therefore, without loss of generality, we assume that the first n client queries are distinct. The total query time for these n queries is:

$$\sum_{k=1}^{\log n} O(2^{k-1} \log n \log \frac{n}{2^k} \log \log \frac{n}{2^k}) .$$

It is simple to verify that this sum is $O(n \log n)$. The following claim concludes the proof of the second part of Theorem 1.

... With probability $1 - 1/n$, `monotonize` computes a monotone function g that is within distance $(2 + O(\delta))\varepsilon$ to f . Each single evaluation of $g(i)$ is computed online in time $O(\log^2 n \log \log n)$, and the amortized query time over the first $m \geq n$ client queries is $O(\log n)$.

3 The $(\log n, O(\log n))$ -Filter

We prove Theorem 2. To do this, we define a function g by a random process. The function is determined after some coin flipping done by the algorithm (before handling the client queries). Although the function g is defined after the coin flips, the algorithm doesn't explicitly know it. In order to explicitly calculate g at a point, the algorithm will have to do some f -lookups. Our construction and analysis will upper bound $\mathbf{E}[\text{dist}(f, g)]$ and the amount of work required for explicitly calculating g at a point.

As before, let \hat{f} be a monotone function such that $\text{dist}(f, \hat{f}) = \varepsilon$. Let $B \subseteq [n]$ be the set of points $\{x | f(x) \neq \hat{f}(x)\}$. So $|B| = \varepsilon n$. For simplicity of notation, assume the formal values of $-\infty$ (resp. $+\infty$) of any function on $[n]$ evaluated at 0 (resp. $n + 1$).

We build a randomized binary tree $T = \text{build-tree}(1, n)$ with nodes labeled $1..n$, where $\text{build-tree}(a, b)$ is defined as follows:

```

build-tree( $a, b$ )

  if  $a > b$  then
    return empty tree
  else
    return tree with
      root  $i$  chosen uniformly at random in  $[a, b]$ 
      left subtree build-tree( $a, i - 1$ )
      right subtree build-tree( $i + 1, b$ )

```

After constructing the randomized tree T , the function g at point i is defined as follows. If i is the root of the tree, then $g(i) = f(i)$. Otherwise, Let p_1, \dots, p_j, i

denote the labels of the nodes on the path from the root to node i , where p_1 is the root of the tree and p_j is the parent of i . Assume that g was already defined on p_1, \dots, p_j . Let $l = \max(\{0\} \cup \{p_k | p_k < i\})$ and $r = \min(\{n+1\} \cup \{p_k | p_k > i\})$. If $g(l) \leq f(i) \leq g(r)$, then define $g(i) = f(i)$, otherwise define $g(i)$ as an arbitrary value in $[g(l), g(r)]$. The function g is clearly monotone. The number of f -lookups required for computing $g(i)$ is the length of the path from the root to i . We omit the proof of the following fact.

Lemma 6. *For any $i \in [n]$, $\mathbf{E}[\text{dist}(f, g)] = O(\log n)$.*

We show that $\mathbf{E}[\text{dist}(f, g)] = O(\varepsilon \log n)$. We first observe that for any i , if $\{p_1, \dots, p_j, i\} \cap B = \emptyset$, then it is guaranteed that $g(i) = f(i)$. Therefore, any i for which $f(i) \neq g(i)$ can be charged to some $b \in B$ on the path from the root to i . The amount of charge on any $b \in B$ is at most the size of the subtree b in T . We omit the proof of the following lemma.

Lemma 7. *For any $i \in [n]$, $\mathbf{E}[\text{dist}(f, g)] = O(\log n)$.*

Therefore, the expected total amount of charge is at most $O(|B| \log n) = O(n\varepsilon \log n)$. By Chebyshev’s inequality, the total amount of charge is at most $O(n\varepsilon \log n)$ with high probability. The total amount of charge is an upper bound on the distance between f and g . This proves Theorem 2, except for the fact that the lookups-per-query guarantee is only on expectation, and not worst case (due to Lemma 6). We note without proof that the random tree T can be constructed as a balanced binary tree, so that Lemma 6 is unnecessary, and Lemma 7 is still true. So we get a worst case (instead of expected) guarantee of $O(\log n)$ on the length of the path from the root to i (and hence on the number of f -lookups per client query). This concludes the proof of Theorem 2.

To prove Theorem 3, where the client queries are assumed to be uniformly and independently chosen in $[n]$, we observe that the choices the client makes can be used to build T . More precisely, we can build T on the fly, as follows: The root r of T is the first client query. The left child of r is the first client query in the interval $[1, r - 1]$, and the right child of r is the first client query in the interval $[r + 1, n]$. In general, the root of any subtree in T is the first client query in the interval corresponding to that subtree. Clearly, this results in a tree T drawn from the same probability distribution as in `build-tree(1, n)`. So we still have Lemma 7, guaranteeing the upper bound on the expected distance between g and f . But now we observe that for any new client query i , the path from the root of T to i (excluding i) was already queried, so we need only one more f -lookup, namely $f(i)$. This concludes the proof of Theorem 3. \square

References

1. Ailon, N., Chazelle, B., Comandur, S., Liu, D.: Estimating the distance to a monotone Function. Proc. 8th RANDOM, 2004

2. Fischer, E.: The art of uninformed decisions: A primer to property testing. *Bulletin of EATCS*, 75: 97-126, 2001
3. Goldreich, O.: Combinatorial property testing - A survey. "Randomization Methods in Algorithm Design," 45-60, 1998
4. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. *J. ACM* 45 (1998), 653-750
5. Hoffmann, C.M., Hopcroft, J.E., Karasick, M.S.: Towards implementing robust geometric computations. *Proc. 4th SOCG (1988)*, 106-117.
6. Parnas, M., Ron, D., Rubinfeld, R.: Tolerant property testing and distance approximation. *ECCC 2004*
7. Ron, D.: Property testing. "Handbook on Randomization," Volume II, 597-649, 2001
8. Rubinfeld, R., Sudan, M.: Robust characterization of polynomials with applications to program testing. *SIAM J. Comput.* 25 (1996), 647-668
9. Salesin, S., Stolfi, J., Guibas, L.J.: Epsilon geometry: building robust algorithms from imprecise computations. *Proc. 5th SOCG (1988)*, 208-217

On the Monotone Circuit Complexity of Quadratic Boolean Functions

Kazuyuki Amano and Akira Maruoka

Graduate School of Information Sciences, Tohoku University,
Aoba 6-6-05, Aramaki, Sendai 980-8579, Japan
{ama|maruoka}@ecei.tohoku.ac.jp

Abstract. Several results on the monotone circuit complexity and the conjunctive complexity, i.e., the minimal number of AND gates in monotone circuits, of quadratic Boolean functions are proved. We focus on the comparison between single level circuits, which have only one level of AND gates, and arbitrary monotone circuits, and show that there is a huge gap between the conjunctive complexity of single level circuits and that of general monotone circuits for some explicit quadratic function. Almost tight upper bounds on the largest gap between the single level conjunctive complexity and the general conjunctive complexity over all quadratic functions are also proved. Moreover, we describe the way of lower bounding the single level circuit complexity, and give a set of quadratic functions whose monotone complexity is strictly smaller than its single level complexity.

1 Introduction

Deriving a superlinear lower bound on the Boolean circuit-size complexity for an explicit Boolean function is one of the most challenging problems in computational complexity. In order to attack the problem, the complexity of many types of restricted circuits have been investigated. The model of monotone Boolean circuits, i.e., circuits with only AND and OR gates, is one of the most well-studied models.

In this paper, we investigate the monotone circuit complexity of the class of quadratic Boolean functions, i.e., functions of the form $\bigvee_{i,j} a_{i,j} \wedge x_i \wedge x_j$ where $a_{i,j} \in \{0, 1\}$. Although we have a series of strong lower bounds on the monotone circuit complexity of explicitly defined Boolean functions [2, 3, 4, 5, 7, 8, 9, 15, 16, 18], such as exponential lower bounds for the clique function, we believe that an investigation of the monotone complexity of quadratic functions is important for several reasons:

(i) The method of approximations and many variants of them have been successful to obtain exponential lower bounds on the monotone circuit complexity [2, 3, 4, 5, 7, 8, 9, 15, 16, 18]. However, several researchers have pointed out that these methods are shown to be equivalent [4, 5, 8, 9, 18]. In addition, a simple analysis of the method shows that it cannot yield any non-trivial lower

bounds on the circuit complexity of a quadratic Boolean function. This is because the method is in fact lower bounding the number of AND gates and that of OR gates needed to compute the function, and every quadratic Boolean function on n variables can be computed by a monotone circuit including at most $n - 1$ AND gates. So a superlinear lower bound for quadratic Boolean functions may imply an essentially different method for lower bounding the monotone circuit complexity. (ii) For some natural class of quadratic Boolean functions, which we will describe in Section 4, we can show that a superlinear lower bound on the monotone circuit complexity of a function f in that class immediately implies the lower bound of the same order on the circuit complexity of f . In addition, we hope that a lower bound proof that is highly specialized for a particular function may not fulfill the “largeness” condition in the notion of “natural proof” [17].

A quadratic Boolean function is naturally represented by a graph. Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \dots, n\}$ and edge set $E \subseteq \{(i, j) \mid 1 \leq i < j \leq n\}$. A quadratic (Boolean) function associated with G is defined by $f_G(x_1, \dots, x_n) = \bigvee_{(i,j) \in E} x_i x_j$.

There have been a series of researches on the complexity of quadratic Boolean functions (sometimes under the name of graph complexity), which are mostly concerned on the circuits of constant depth with unbound fan-in gates (e.g., [10, 12, 14]). In this paper, we restrict the fan-in of gates to two and mainly focus on the comparison between single level monotone circuits and general monotone circuits. A single level circuit is a circuit which has only one level of AND gates. Obviously, every quadratic function can be computed by a single level circuit of size $O(n^2)$. Not surprisingly, if we restrict ourselves to circuits of single level, we can easily derive a superlinear lower bound on the size. (We will describe this in more detail in Section 3.3.) However, it seems quite difficult to obtain a good lower bound on the general monotone circuit size for an explicit quadratic function.

One of the major difference between single level circuits and general circuits in a computation of quadratic functions relies on the use of “absorption” rule, i.e., $f \vee fg = f$ (and $f(f \vee g) = f$). We think that an investigation on the efficiency of the absorption rule in a monotone computation may be a key to obtain a tighter/higher lower bound on the monotone complexity, and this was the initial motivation of our work.

The contributions of this paper are as follows: First, in Section 3, we consider the conjunctive complexity of quadratic Boolean functions. The conjunctive complexity of a quadratic function f_G is the minimal number of AND gates in a monotone circuit that computes f_G . Such measures have been widely studied by e.g., Tuza [20], Lenz and Wegener [11]. In Section 3.2, we prove that there is a huge gap between the conjunctive complexity of single level circuits and that of general monotone circuits for some explicit quadratic function (Theorem 2). Almost tight upper bounds on the largest gap between the single level conjunctive complexity and the general conjunctive complexity over all quadratic

functions are also proved (Theorem 4). Then, in Section 3.3, we describe the way of lower bounding the single level circuit complexity (Theorem 8), and give a set of quadratic functions whose monotone circuit complexity is strictly smaller than its single level complexity (Theorem 10). Finally, in Section 4, we discuss the relationship between the complexity of monotone circuits and of non-monotone circuits for quadratic functions based on the notion of pseudo complements (Theorem 11).

2 Boolean Circuits

A \dots is a directed acyclic graph. Nodes with indegree zero are called \dots nodes and there are distinguished nodes with outdegree zero called \dots nodes. An input node is labeled by a Boolean variable or a constant 0 or 1. Each non-input node has indegree 2 or 1 and is called the \dots node. A gate node of indegree 2 is labeled by a Boolean operation AND (\wedge) or OR (\vee). A gate node of indegree 1 is labeled by a Boolean operation NOT (\neg). A gate in a Boolean circuit computes a Boolean function in a natural way. If g is a gate in a Boolean circuit, we will also use g to denote the function computed by g . A Boolean circuit computes Boolean functions that are computed by the output gates. A \dots is a Boolean circuit that contains no NOT gates. A Boolean function that can be computed by a monotone Boolean circuit is called a \dots . Since we will not discuss any non-Boolean functions, we may drop the word ‘‘Boolean’’.

3 Single Level Versus Multi Level

3.1 Notations

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \dots, n\}$ and edge set $E \subseteq \{(i, j) \mid 1 \leq i < j \leq n\}$. A quadratic (Boolean) function associated with G is defined by $f_G(x_1, \dots, x_n) = \bigvee_{(i,j) \in E} x_i x_j$.

For a monotone circuit C , the \dots of C is defined as the maximal number of AND gates on a path from an input to an output in C . In particular, a circuit of level 1, i.e., a circuit such that no path combines AND gates, is called a \dots circuit. Obviously, for every graph $G = (V, E)$ on $V = \{x_1, \dots, x_n\}$, the function f_G can be computed by a single level circuit including at most $|V| - 1$ AND gates and $|E|$ OR gates using the form $\bigvee_{1 \leq i < n} x_i \wedge (\bigvee_{j:(i,j) \in E} x_j)$.

Let f be a monotone function. The \dots (the \dots , resp.) of f , denoted by $\dots(f)$ ($\dots_{mon}(f)$, resp.), is the minimal number of gates in a Boolean circuit (a monotone circuit, resp.) for f , and the single level complexity of f , denoted by $\dots^1_{mon}(f)$, is the minimal number of gates in a single level monotone circuit for f .

In this paper, we also investigate the number of AND gates needed to compute a function, which is called as the \dots . The conjunc-

tive complexity (the single level conjunctive complexity, resp.) of f , denoted by $\dots_{mon,\wedge}(f)$ ($\dots^1_{mon,\wedge}(f)$, resp.), is the minimal number of AND gates in a monotone circuit (a single level monotone circuit, resp.) that computes f .

3.2 Conjunctive Complexity

If we restrict ourselves to circuits of single level, the conjunctive complexity of a quadratic function f_G is equal to the minimal number of complete bipartite graphs whose union (of the edge sets) coincides with G .

Since the single level conjunctive complexity $\dots^1_{mon,\wedge}(f)$ is a purely graph theoretical complexity measure, it has been widely studied (see e.g., [11]). In contrast, little is known about the conjunctive complexity $\dots_{mon,\wedge}(f)$. In the following, we show that almost all quadratic functions have a conjunctive complexity $\Theta(n)$, which improves the $n/(c \log n)$ lower bound of Lenz and Wegener [11]. We remark that it was shown that almost all quadratic functions have a single level conjunctive complexity of larger than $n - c \log n$.

Theorem 1. *For almost all quadratic functions f_G on n vertices, $\dots_{mon,\wedge}(f_G) \geq cn$ for $c < 1/13$.*

The proof of the theorem is analogous to the proof of the $n/(c \log n)$ lower bound due to Lenz and Wegener [11]. The only difference is to use the result of Zwick [24], which says that if f can be computed by a monotone circuit that contains k AND gates, then f can also be computed by a monotone circuit that contains k AND gates and $O(k(n+k)/\log k)$ OR gates, instead of the result of Alon and Boppana [2], which is slightly weaker than the result of Zwick.

Careful inspection of Zwick’s proof reveals that the hidden constant in the Big-O notation of their upper bound is at most 3. The standard counting argument shows that, for each $d > 4$, the monotone circuit complexity of almost all quadratic functions is at least $n^2/(d \log n)$. Hence, for almost all graphs on n vertices G , if we denote the conjunctive complexity of f_G by k , then

$$\frac{n^2}{d \log n} \leq \frac{3k(n+k)}{\log k} + k.$$

holds. A simple calculation shows that $k \geq cn$ for sufficiently large n . □

One might conjecture that an optimal circuit for a quadratic function with respect to the conjunctive complexity is always given by a single level circuit. This was known as *single level conjecture* (with respect to the conjunctive complexity) and was *proved* by Lenz and Wegener [11]. They provided an explicit graph H on 8 vertices such that $4 = \dots^1_{mon,\wedge}(f_H) > \dots_{mon,\wedge}(f_H) = 3$, and asked what the largest possible value of $\dots^1_{mon,\wedge}(f_G)/\dots_{mon,\wedge}(f_G)$ is (as open problem No. 7 in [11]).

In the following, we improve their result by giving an explicit construction of a graph G on n vertices such that $\dots^1_{mon,\wedge}(f_G)/\dots_{mon,\wedge}(f_G) = \Omega(n/\log n)$.

Theorem 2. $f_G \in \text{mon}, \wedge^1(f_G) = \Omega(n)$
 $\text{mon}, \wedge^1(f_G) = O(\log n)$

Let $\tilde{G} = (U \cup V, \tilde{E})$ be a bipartite graph with $U = \{u_1, \dots, u_{n/2}\}$, $V = \{v_1, \dots, v_{n/2}\}$ and $\tilde{E} = \{(u_i, v_i) \mid i = 1, \dots, n/2\}$. For simplicity, we assume that $n = 2^t$ for some positive integer t . Let G be a graph on $U \cup V$ such that $G = \tilde{G} \cup K_U \cup K_V$ where K_U and K_V are the complete graphs on U and V respectively. In the following, we show that $\text{mon}, \wedge^1(f_G) \geq n/4$ and $\text{mon}, \wedge^1(f_G) = O(\log n)$.

First, we show that $\text{mon}, \wedge^1(f_G) \geq n/4$. Let C be a single level monotone circuit for f_G . For a function g , let $PI(g)$ denote the set of all prime implicants of g . Since \tilde{G} contains $n/2$ edges, it is sufficient to show that for every \wedge gate g in C , if $PI(g)$ contains more than two edges in \tilde{G} then g can be eliminated without changing the output of C . Let g_1 and g_2 be two inputs of g . Note that g_1 and g_2 are OR's of variables. Suppose that $PI(g)$ contains three edges in \tilde{G} , say $(u_i, v_i), (u_j, v_j), (u_k, v_k)$. Then, w.l.o.g., we can assume that for some distinct indices $i_1, i_2 \in \{i, j, k\}$, u_{i_1} and u_{i_2} are appearing in g_1 and v_{i_1} and v_{i_2} are appearing in g_2 . This implies that $PI(g)$ contains (u_{i_1}, v_{i_2}) , which is not included in \tilde{G} . Hence g cannot contribute the output of C , and can be removed safely.

We now show that $\text{mon}, \wedge^1(f_G) = O(\log n)$. Let d be a positive integer whose value will be chosen later. Let $l = 2^{d-1}$ and $r = (n/2)^{1/l}$. For simplicity, we assume that r is an integer. For $1 \leq k \leq n/2$, we represent k by a vector $k = (k_1, \dots, k_l) \in \{1, \dots, r\}^l$. It will be convenient to consider that k is represented by an r -ary l -digits number. For $1 \leq i \leq l$ and $1 \leq j \leq r$, let P_j^i (Q_j^i , resp.) be the set of n/r variables consists of all u_k (v_k , resp.) such that $k = (r_1, \dots, r_{i-1}, j, r_{i+1}, \dots, r_l)$ for some $r_1, \dots, r_l \in \{1, \dots, r\}$, i.e., the i -th digit of the r -ary representation of k is equal to j .

We claim that f_G is equivalent to

$$\bigwedge_{1 \leq i \leq l} \left(\bigvee_{1 \leq j \leq r} (OR(P_j^i) \wedge OR(Q_j^i)) \right) \vee \left(Th_2^{n/2}(U) \vee Th_2^{n/2}(V) \right), \quad (1)$$

where $OR(X)$ denotes the disjunction of all variables of X and $Th_k^n(X)$ denotes the k -threshold function on n variables, i.e., it outputs 1 iff the number of ones in an input is greater than or equal to k .

Before we show the correctness of Eq. (1), we determine the value of d and estimate the number of AND gates needed to compute Eq. (1). Since (i) the AND of l functions can be computed by a circuit of level $\log l = d - 1$ with $l - 1$ AND gates, and (ii) the 2-threshold function on $n/2$ variables can be computed by a single level circuit that includes $\log(n/2) = \log n - 1$ AND gates¹, we can construct a d -level circuit including at most

¹ Proof: Let $X = \{x_1, \dots, x_n\}$ and consider that an integer $k = 1, \dots, n$ is represented by a $\log n$ -digits binary number. Let $X_{i,j}$ ($j \in \{0, 1\}$) be the OR of all x_k such that i -th digit of binary representation of k is equal to j . Then it is easy to check that $Th_2^n \equiv \bigvee_{1 \leq i \leq \log n} (X_{i,0} \wedge X_{i,1})$.

$$lr + l - 1 + 2(\log n - 1) < 2^{d-1}((n/2)^{1/2^{d-1}} + 1) + 2 \log n \quad (2)$$

AND gates. If we choose $d = \log \log n$, the RHS of Eq. (2) is upper bounded by $4.5 \log n$.

Now we proceed to the proof of the correctness of Eq. (1). Obviously, for an input with at most 1 ones, both f_G and Eq. (1) output 0. For an input with at least 3 ones, both f_G and Eq. (1) output 1 because (at least) one of two sets U and V contain at least 2 variables that assigned the value 1. Thus, the interesting cases are for an input with 2 ones.

(Case 1) $u_{k_1} = u_{k_2} = 1$ or $v_{k_1} = v_{k_2} = 1$ for some $k_1 \neq k_2$.

Obviously, both f_G and Eq.(1) output 1 in this case.

(Case 2) $u_k = v_k = 1$ for some $k \in \{1, \dots, n/2\}$.

By the definition of G , f_G outputs 1 on such an input. For each $1 \leq i \leq l$, $OR(P_{k_i}^i) = OR(Q_{k_i}^i) = 1$ if k_i is equal to the i -th digit of r -ary representation of k . This implies the output of Eq. (1) is also 1.

(Case 3) $u_{k_1} = v_{k_2} = 1$ for some $k_1 \neq k_2$.

By the definition of G , f_G outputs 0 on such an input. Since $k_1 \neq k_2$, there is $1 \leq i \leq l$ such that the i -th digit of the r -ary representation of k_1 and k_2 are different. For such i , the value of $\bigvee_{1 \leq j \leq l} (OR(P_j^i) \wedge OR(Q_j^i))$ is 0, and this implies the output of Eq. (1) is also 0. \square

The graph we defined in the proof of Theorem 2 consists of three sub-graphs, \tilde{G} , K_U and K_V . Interestingly, the single level conjunctive complexity and the conjunctive complexity of each subgraph are identical, i.e., $\dots \frac{1}{mon, \wedge}(\tilde{G}) = \dots \frac{1}{mon, \wedge}(\tilde{G}) = n/2$, $\dots \frac{1}{mon, \wedge}(K_U) = \dots \frac{1}{mon, \wedge}(K_U) = \log n - 1$ and $\dots \frac{1}{mon, \wedge}(K_V) = \dots \frac{1}{mon, \wedge}(K_V) = \log n - 1$.

Let $Gap_{\wedge}(n) = \max\{\dots \frac{1}{mon, \wedge}(f_G) / \dots \frac{1}{mon, \wedge}(f_G) \mid G = (V, E), |V| = n\}$. Theorem 2 shows that $Gap_{\wedge}(n) = \Omega(n/\log n)$. Note that the upper bound of $O(n)$ is trivial since $\dots \frac{1}{mon, \wedge}(f_G) \leq n - 1$ for every G on n vertices. We conjecture that our lower bound is tight, i.e., $Gap_{\wedge}(n) = \Theta(n/\log n)$. Below we prove a slightly weaker upper bound of $Gap_{\wedge}(n) = O(n/\log \log n)$.

Theorem 4. $\dots \frac{1}{mon, \wedge}(f_G) = \Omega(p(n))$ $\dots \frac{1}{mon, \wedge}(f_G) = \Omega(\log \log p(n))$

(Sketch) Let G be a graph on n vertices whose single level conjunctive complexity is $\Omega(p(n))$. Let C be an arbitrary monotone circuit that computes f_G . Below we show that C must contain $\Omega(\log \log p(n))$ AND gates.

For a monotone function h , let $PI_i(h)$ be the set of all prime implicants of h whose length is i . The i -th cover of h , denoted by $cov_i(h)$, is defined as the minimal m such that there are m pairs of sets of variables (A_i, B_i) ($i = 1, \dots, m$) that satisfy (i) $A_i \cap B_i = \emptyset$ ($\forall i$), (ii) $A_i \times B_i \subseteq PI_2(h)$ ($\forall i$), and (iii) $\bigcup_{i=1}^m A_i \times B_i = PI_2(h)$. In such a case, we say that a set of pairs $\{(A_i, B_i) \mid i = 1, \dots, m\}$ $\dots \dots PI_2(h)$. Obviously, $cov(f_G) = \dots \frac{1}{mon, \wedge}(f_G)$ for every G .

We define the operation \wedge^* as follows: Suppose that $g = g_1 \wedge g_2$. Then $g_1 \wedge^* g_2$ is the disjunction of all prime implicants of g whose length is at most 2. Let C^* be a circuit obtained from C by replacing each \wedge gate in C with \wedge^* gate. The theorem known as the “replacement rules” (see e.g., [22, Theorem 5.1]) guarantees that the circuit C^* also computes f_G . In the following, we assume that if g_1 and g_2 are two inputs of an \wedge^* gate in C^* , then $PI_1(g_1)$ and $PI_1(g_2)$ are disjoint. (If $PI_1(g_1) \cap PI_1(g_2) = S \neq \emptyset$, then we replace $g_1 \wedge^* g_2$ by $(h_1 \wedge^* h_2) \vee OR(S)$ where h_i is obtained from g_i by removing all prime implicants in S . This does not affect on the number of \wedge^* gates in C^* and the output of C^* . In addition, this replacement has no influence on the covering number of each gate.)

For an \wedge^* gate g in C^* , the *depth* of g is defined as the maximal number of \wedge^* gates on a path from an input to (the output of) g . Note that the lowest \wedge^* gate is of level 1. Let d be the level of the circuit C^* , and for $i = 1, \dots, d$, let k_i be the number of \wedge^* gates whose level is i . Note that the number of \wedge^* gates in C^* , say k , is given by $k = \sum_{i=1}^d k_i$.

Let $g = g_1 \wedge^* g_2$ be an arbitrary \wedge^* gate of level l in C^* . We claim that

$$cov(g) \leq 5 \cdot 6^{2^{l-1}-2} \max\{k_1, \dots, k_{l-1}\}^{2^l-2}, \quad (\text{for } l \geq 2), \quad (3)$$

$$cov(g_i) \leq 6^{2^{l-2}-1} \max\{k_1, \dots, k_{l-1}\}^{2^{l-1}-1}, \quad (\text{for } l \geq 2 \text{ and } i = 1, 2). \quad (4)$$

To prove these inequalities, we need the following lemma.

Lemma 5. Let $h = h_1 \wedge^* h_2$, $PI_1(h_1) \cap PI_1(h_2) = \emptyset$, $cov(h_1), cov(h_2) \geq 1$. Then $PI_1(h_1) \cap PI_1(h_2) = \emptyset$ and $cov(h) \leq 5 \cdot cov(h_1)cov(h_2)$.

□

Let $m_1 = cov(h_1)$ and $m_2 = cov(h_2)$. We can express h_1 and h_2 as

$$h_1 = t_1 \vee \bigvee_{i=1}^{m_1} a_{i,1} b_{i,1}, \quad h_2 = t_2 \vee \bigvee_{i=1}^{m_2} a_{i,2} b_{i,2},$$

where the t_j , $a_{i,j}$ and $b_{i,j}$ are disjunctions of variables such that $Var(a_{i,j}) \cap Var(b_{i,j}) = \emptyset$ ($\forall i, j$) and $Var(t_1) \cap Var(t_2) = \emptyset$. We have

$$\begin{aligned} PI_2(h_1 \wedge^* h_2) &= \bigvee_{i_1, i_2} PI_2(a_{i_1,1} a_{i_2,1} b_{i_1,1} b_{i_2,2}) \\ &\vee \bigvee_{i_2} PI_2(t_1 a_{i_2,2} b_{i_2,2}) \vee \bigvee_{i_1} PI_2(t_2 a_{i_1,1} b_{i_1,1}) \vee PI_2(t_1 t_2). \end{aligned}$$

It is easy to check that, for each pair (i_1, i_2) , $PI_2(a_{i_1,1} a_{i_2,1} b_{i_1,1} b_{i_2,2})$ can be covered by two pairs $(a_{i_1,1} \cap a_{i_2,2}, b_{i_1,1} \cap b_{i_2,2})$ and $(a_{i_1,1} \cap b_{i_2,2}, a_{i_2,2} \cap b_{i_1,1})$. Similarly, each $PI_2(t_1 a_{i_2,2} b_{i_2,2})$ ($PI_2(t_2 a_{i_1,1} b_{i_1,1})$, resp.) can be covered by a pair $(t_1 \cap a_{i_2,2}, t_1 \cap b_{i_2,2})$ ($(t_2 \cap a_{i_1,1}, t_2 \cap b_{i_1,1})$, resp.). Altogether, $PI_2(h_1 \wedge^* h_2)$ can be covered by a set of at most $2m_1 m_2 + m_1 + m_2 + 1 \leq 5m_1 m_2$ pairs.

□

(.....) By Lemma 5, Eq. (4) immediately implies Eq. (3) for each l . Hence we only need to show Eq. (4). We show this by induction on l .

The base case, $l = 2$, is obvious since RHS of Eq. (4) is k_1 and the covering number of an input of an \wedge^* gate of level 2 is shown to be at most k_1 .

The induction step is as follows : Since the function computed by an input of an \wedge^* gate of level l can be represented by the disjunction of variables and outputs of \wedge^* gates of level at most $l - 1$, the covering number of it is upper bounded by

$$5k_{l-1}6^{2^{l-2}-2} \max\{k_1, \dots, k_{l-2}\}^{2^{l-1}-2} + 6^{2^{l-3}-1} \max\{k_1, \dots, k_{l-2}\}^{2^{l-2}-1} \leq 6^{2^{l-2}-1} \max\{k_1, \dots, k_{l-1}\}^{2^{l-1}-1},$$

which completes the proof of the induction step.

The assumption $\dots \frac{1}{mon, \wedge}(f_G) = \Omega(p(n))$ in the statement of the theorem implies that k times the value of Eq. (3) for $l = d$ is $\Omega(p(n))$, and this implies $(6k)^{2^k} \geq (6k)^{2^d} = \Omega(p(n))$ (since $k \geq d$). Hence we have $k = \Omega(\log \log p(n))$, which completes the proof of the theorem. □

3.3 Disproving Single Level Conjecture for Multi-output Functions

Again, if we restrict ourselves to circuits of single level, a good lower bound on $\dots \frac{1}{mon}(f)$ can easily be derived by combining the graph theoretic arguments and the results that have been developed for obtaining a lower bound on the monotone complexity of the \dots , which we state below.

Definition 6. $\dots X = \{x_1, \dots, x_n\}$ $F(X) \equiv (f_1, \dots, f_m) : \{0, 1\}^n \rightarrow \{0, 1\}^m$
 $\dots f_i \dots X$
 $\dots f \dots X$
 $\dots Var(f) \dots X$
 $\dots F \dots (h, k)$
 $\dots \{i_0, \dots, i_h\} \subseteq \{1, \dots, m\} \dots |\bigcup_{i=0}^h Var(f_i)| \leq k \dots$

Theorem 7 ([13]). $\dots F = (f_1, \dots, f_m) \dots F$
 $(h, k) \dots F$

$$\sum_{1 \leq i \leq m} \frac{[|Var(f_i)|/k] - 1}{h \max\{1, h - 1\}}.$$

By using the above theorem, we can show the following.

Theorem 8. $\dots G = (V, E) \dots K_{2,2}$
 $\dots \frac{1}{mon}(f_G) \geq |E|$

Let C be an optimal single level monotone circuit for f_G . We represent $f_G = \bigvee_{i=1}^k g_{i,1}g_{i,2}$, where k is the number of AND gates in C , and $g_{i,1}$ and $g_{i,2}$ are disjunctions of variables computed by the inputs of the i -th AND gate.

For each $i = 1, \dots, k$, at least one of $Var(g_{i,1})$ or $Var(g_{i,2})$ contain at most one variable. This is because if $|Var(g_{i,1})| \geq 2$ and $|Var(g_{i,2})| \geq 2$, then G must contain $K_{2,2}$ (if two sets are disjoint), or f_G must contain a prime implicant of length 1 (if two sets are not disjoint). Hence, without loss of generality, we can assume that $|Var(g_{i,1})| = 1$ for every $i = 1, \dots, k$ (by exchanging $g_{i,1}$ and $g_{i,2}$ if necessary). Let $X = \bigcup_i Var(g_{i,1})$. Now we convert C to a circuit C' by the following construction:

$$\bigvee_{x \in X} x \wedge \left(\bigvee_{j:Var(g_{j,1})=\{x\}} g_{j,2} \right).$$

Since we can save $k - |X|$ AND gates and the number of additional OR gates we need is shown to be at most $k - |X|$, the size of C' is not greater than that of C .

For each $x \in X$, let $h_x = \bigvee_{j:Var(g_{j,1})=\{x\}} g_{j,2}$. It is obvious that $Var(h_x)$ is a subset of the set of neighbors of x . Since G does not contain $K_{2,2}$, $|Var(h_{x_1}) \cap Var(h_{x_2})| \leq 1$ for every distinct $x_1, x_2 \in X$. Hence the set of functions $H = \{h_x \mid x \in X\}$ can be viewed as the $(1, 1)$ -disjoint Boolean sums. Therefore, the size of C' is at least

$$\dots \text{mon}(H) + |X| + |X| - 1 \geq |E| - |X| + 2|X| - 1 \geq |E|.$$

The first inequality follows from Lemma 7. □

An explicit construction of the graph on n vertices that does not contain $K_{2,2}$ and has $\Omega(n^{3/2})$ edges based on the notion of the “projective plane” was known (e.g., [1]). The above theorem yields $\dots \frac{1}{\text{mon}}(f_G) = \Omega(n^{3/2})$ for such G . We remark that we can extend the arguments of the proof of Theorem 8 for a graph that does not contain a copy of $K_{r,r}$ for $r > 2$. Thus an explicit construction for such a graph may yield higher lower bounds on the size of single level circuits.

The question that then arises is : “Is there a quadratic function f such that $\dots \text{mon}(f_G)$ is strictly smaller than $\dots \frac{1}{\text{mon}}(f_G)$?”

The problem of answering this question was stated as open problem in [11]. We have shown in the previous section that the answer is “yes” if we only count the number of AND gates. In the following, we show that the answer is also “yes” if we consider \dots quadratic functions.

For a set of m graphs $\mathcal{G} = (G_1, \dots, G_m)$, a set of quadratic functions associated with \mathcal{G} , denoted by $f_{\mathcal{G}}$, is defined by the set of m functions $(f_{G_1}, \dots, f_{G_m})$.

Lemma 9. $\dots H = (h_1, \dots, h_m) \dots \{x_1, \dots, x_n\} \dots F = (f_1, \dots, f_m) \dots U \cup V = \{u_1, \dots, u_n\} \cup$

4 Monotone Versus Non-monotone

The graph G that we defined in the proof of Theorem 2 has the form of $G = \tilde{G} \cup K_U \cup K_V$ where $\tilde{G} \subseteq U \times V$ is a bipartite graph, and K_U and K_V are the complete graphs on U and V respectively. Interestingly, it is shown that NOT gates are almost powerless for quadratic functions associated with graphs of such form.

Theorem 11. (Wegener [21]) Let $G = \tilde{G} \cup K_U \cup K_V$ where $\tilde{G} \subseteq U \times V$ is a bipartite graph, and $U, V \subseteq \{1, \dots, n\}$ are disjoint sets. Then, \dots $mon(f_G) \leq 2 \dots (f_G) + 6n + o(n)$.

(Sketch) In fact, the result by Wegener [21, Theorem 4.4] is of the form $\dots mon(f_G) \leq O(\dots (f_G)) + O(n)$. For the purpose of completeness and in order to determine the hidden constants, we describe the sketch of the proof.

Given an optimal circuit C computing f_G . By using the DeMorgan's law, we can convert C to a so-called standard circuit C' for f_G , that is a Boolean circuit in which the permitted gate operations are $\{\wedge, \vee\}$ and whose input nodes are labeled by literals, whose size is at most twice of the size of C .

Let $U = \{u_1, \dots, u_k\}$ and $V = \{v_1, \dots, v_{n-k}\}$. Let $u'_i = u_i \wedge OR(V)$, $v'_i = v_i \wedge OR(U)$, $\tilde{u}_i = \vee_{j \neq i} u'_j$ and $\tilde{v}_i = \vee_{j \neq i} v'_j$. Let f' be a function computed by a circuit obtained from C' by replacing each u_i (v_i , resp.) with u'_i (v'_i , resp.) and each \bar{u}_i (\bar{v}_i , resp.) with \tilde{u}_i (\tilde{v}_i , resp.). The key to the proof is the observation that f_G can be computed by $f' \vee Th_2^{|U|}(U) \vee Th_2^{|V|}(V)$. (In other words, we can use u'_i and v'_i as \dots and \tilde{u}_i and \tilde{v}_i as \dots for u_i and v_i respectively. See [21] for more details.)

By following the equation

$$Th_2^n(X) = \bigvee_{q=1}^2 Th_2^{\sqrt{n}} \left(OR(B_1^q), \dots, OR(B_{\sqrt{n}}^q) \right),$$

where $X = \{x_{r_1 r_2} \mid 1 \leq r_1, r_2 \leq \sqrt{n}\}$ and $B_i^q = \{x_{s_1 s_2} \mid s_q = i\}$, we can compute $Th_2^{|U|}$ and $Th_2^{|V|}$ with at most $2(|U| + |V|) + o(n) = 2n + o(n)$ gates. Clearly, $\sqrt{|U|} + \sqrt{|V|} = o(n)$ additional gates are suffice to compute $OR(U)$ and $OR(V)$. All u'_i and v'_i can be computed with n gates. Moreover, $3|U| + 3|V| = 3n$ gates are suffice to compute all \tilde{u}_i and \tilde{v}_i . Altogether we use at most $6n + o(n)$ gates. \square

We remark that the standard counting argument shows that the circuit complexity of almost all quadratic functions associated with graphs of the form $G \cup K_U \cup K_V$, where $\tilde{G} \subseteq U \times V$, is $\Omega(n^2 / \log n)$.

For a bipartite graph $G \subseteq U \times V$, let G^+ be the graph $G \cup K_U \cup K_V$. The relationship between the monotone complexity of f_G and of f_{G^+} seems to be an interesting. Since $\dots mon(Th_2^n)$ is known to be $2n + o(n)$, we have $\dots mon(f_{G^+}) \leq \dots mon(f_G) + 2n + o(n)$. On the other hand, we do not know whether $\dots mon(f_{G^+}) = \Omega(\dots mon(f_G))$ or not. However, if we consider multi-output functions, computing a set of functions $f_{G_1^+}, \dots, f_{G_m^+}$ may significantly easier than computing a set of functions f_{G_1}, \dots, f_{G_m} .

The n -point Boolean convolution $\text{CONV}_n(x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1})$: $\{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n-1}$ is the function with output (S_0, \dots, S_{2n-2}) defined by

$$S_k(x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}) = \bigvee_{i+j=k} x_i y_i.$$

Each S_k is naturally represented by a bipartite graph $\tilde{G}_k \subseteq U \times V$ where $U = \{x_0, \dots, x_{n-1}\}$ and $V = \{y_0, \dots, y_{n-1}\}$. Let CONV_n^+ denote the function with output $(f_{G_0^+}, \dots, f_{G_{2n-2}^+})$. It was known that the monotone complexity of the n -point Boolean convolution is $\Omega(n^{1.5})$ [23] and the (general) circuit complexity of it is $O(n \log^2 n \log \log n)$ [22, p.168]. These imply that $\dots_{mon}(\text{CONV}_n) = \Omega(n^{1.5})$ whereas $\dots_{mon}(\text{CONV}_n^+) = O(\dots(\text{CONV}_n^+)) + O(n) = O(\dots(\text{CONV}_n) + O(n)) = O(n \log^2 n \log \log n)$.

Acknowledgments. The authors would like to thank anonymous reviewers for their helpful suggestions which helped in improving the quality of this paper.

References

1. N. Alon, Eigenvalues, Geometric Expanders, Sorting in Rounds, and Ramsey Theory, *Combinatorica* **6** (1986) 207–219
2. N. Alon and R. Boppana, The Monotone Circuit Complexity of Boolean Functions, *Combinatorica* **7(1)** (1987) 1–22
3. A. Andreev, On a Method for Obtaining Lower Bounds for the Complexity of Individual Monotone Functions, *Soviet Math. Dokl.* **31(3)** (1985) 530–534
4. K. Amano and A. Maruoka, The Potential of the Approximation Method, *SIAM J. Comput.* **33(2)** (2004) 433–447 (Preliminary version in : *Proc. of 37th FOCS* (1996) 431–440)
5. C. Berg, S. Ulfberg, Symmetric Approximation Arguments for Monotone Lower Bounds Without Sunflowers, *Computational Complexity* **8(1)** (1999) 1–20
6. P.E. Dunne, *The Complexity of Boolean Networks*, Academic Press (1988)
7. A. Haken, Counting Bottlenecks to Show Monotone $P \neq \text{NP}$, *Proc. of 36th FOCS* (1995) 36–40
8. D. Harnik, R. Raz, Higher Lower Bounds for Monotone Size, *Proc. of 32nd STOC* (2000) 191–201
9. S. Jukna, Combinatorics of Monotone Computations, *Combinatorica* **19(1)** (1999) 65–85
10. S. Jukna, On Graph Complexity, *ECCC TR04-004* (2004)
11. K. Lenz and I. Wegener, The Conjunctive Complexity of Quadratic Boolean Functions, *Theor. Comput. Sci.* **81** (1991) 257–268
12. S.V. Lokam, Graph Complexity and Slice Functions, *Theory Comput. Syst.* **36** (2003) 71–88
13. K. Mehlhorn, Some Remarks on Boolean Sums, *Acta Inf.* **12** (1979) 371–375
14. P. Pudlák, V. Rödl and P. Savický, Graph Complexity, *Acta Inf.* **25** (1988) 515–535
15. A. Razborov, Lower Bounds on the Monotone Complexity of Some Boolean Function, *Soviet Math. Dokl.* **31** (1985) 354–357
16. A. Razborov, On the Method of Approximation, *Proc. 21th STOC* (1989) 167–176

17. A. Razborov, S. Rudich, Natural Proofs, *J. Comput. Syst. Sci.* **55(1)** (1997) 24–35
18. J. Simon and S.C. Tsai, On the Bottleneck Counting Argument, *Theor. Comput. Sci.* **237(1-2)** (2000) 429–437
19. R. Tarjan, Complexity of Monotone Networks for Computing Conjunctions, *Ann. Disc. Math.* **2** (1978) 121–133
20. Z. Tuza, Covering of Graphs by Complete Bipartite Subgraphs, Complexity of 0-1 Martix, *Combinatorica* **4** (1984) 111–116
21. I. Wegener, More on the Complexity of Slice Functions, *Theor. Comput. Sci.* **43** (1986) 201–211
22. I. Wegener, *The Complexity of Boolean Functions*, Wiley-Teubner (1987)
23. J. Weiss, An $\Omega(n^{3/2})$ Lower Bound on the Complexity of Boolean Convolution, *Info. and Cont.* **59** (1983) 84–88
24. U. Zwick, On the Number of ANDs versus the Number of ORs in Monotone Boolean Circuits, *Inf. Process. Let.* **59** (1996) 29–30

Generalized Function Matching

Amihood Amir^{1,*} and Igor Nor^{2,**}

¹ Department of Computer Science, Bar-Ilan University,
Ramat-Gan 52900, Israel
+972-3-531-8770, Fax +972-3-736-0498
College of Computing, Georgia Tech, Atlanta, GA 30332-0280
² Department of Computer Science, Bar-Ilan University,
Ramat-Gan 52900, Israel
{amir, norigo}@cs.biu.ac.il

Abstract. We present problems in different application areas: tandem repeats (computational biology), poetry and music analysis, and author validation, that require a more sophisticated pattern matching model that hitherto considered.

We introduce a new matching criterion – *generalized function matching* – that encapsulates the notion suggested by the above problems. The *generalized function matching problem* has as its input a text T of length n over alphabet $\Sigma_T \cup \{\phi\}$ and a pattern $P = P[0]P[1] \cdots P[m-1]$ of length m over alphabet $\Sigma_P \cup \{\phi\}$. We seek all text locations i where the prefix of the substring that starts at i is equal to $f(P[0])f(P[1]) \cdots f(P[m-1])$, for some function $f : \Sigma_P \rightarrow \Sigma_T^*$.

We give a polynomial time algorithm for the generalized pattern matching problem over bounded alphabets. We identify in this problem an important new phenomenon in pattern matching. One where there is a significant complexity difference between the *bounded alphabet* and *infinite alphabet* case. We prove that the generalized pattern matching problem over infinite alphabets is \mathcal{NP} -hard. To our knowledge, this is the first case in the literature where a pattern matching problem over a bounded alphabet can be solved in polynomial time but the infinite alphabet version is \mathcal{NP} -hard.

Keywords: Pattern matching, function matching, parameterized matching, \mathcal{NP} -hard.

1 Introduction

The last few decades have prompted the evolution of pattern matching from a combinatorial solution of the exact string matching problem [15, 17] to an area concerned with approximate matching of various relationships motivated by computational molecular biology, computer vision, and complex searches in

* Partly supported by NSF grant CCR-01-04494 and ISF grant 282/01.

** Partly supported by ISF grant 282/01.

digitized and distributed multimedia libraries [14, 8]. In [2] a new generalized matching paradigm, that of *parameterized matching*, was introduced. In this paper we generalize the function matching model to make it more suitable for several important applications in a number of diverse areas.

In the traditional pattern matching model, one seeks exact occurrences of a given pattern in a text, i.e. text locations where every text symbol is equal to its corresponding pattern symbol. In the *parameterized matching* problem, introduced by Baker [10], one seeks text locations where there exists a bijection f on the alphabet for which every text symbol is equal to the corresponding symbol f of the corresponding pattern symbol. The *parameterized matching* problem was introduced by Baker [10]. Her main motivation lay in software maintenance, where program fragments are to be considered “identical” even if variable names are different. Therefore, strings under this model are comprised of symbols from two disjoint sets Σ and Π containing fixed symbols and parameter symbols respectively. In this paradigm, one seeks occurrences of a string in another, i.e., occurrences up to renaming of the parameter symbols, of a string in another. This renaming is a bijection $b : \Pi \rightarrow \Pi$.

In [2] it was pointed out that for some applications f cannot be a bijection. Rather, it should be simply a function. More precisely, P matches T at location i if for every element $a \in \Sigma$, occurrences of a have the same corresponding symbol in T . In other words, unlike in parameterized matching, there may be a

different symbols in the pattern which are mapped to the same text symbol.

An example of a problem where parameterized matching proves insufficient and function matching is required is the following.

One of the interesting problems in web searching is searching for color images (e.g. [20, 9, 5]). The simplest possible cases is searching for an icon in a screen, a task that the Human-Computer Interaction Lab at the University of Maryland was confronted with. If the colors were fixed, then this is exact two-dimensional pattern matching [4]. However, if the color map is different the exact matching algorithm would not find the pattern. Parameterized two dimensional search is precisely what is needed. If, in addition, we are also willing to lose resolution, then we would use a two dimensional function matching search.

However, even a function $f : \Sigma \rightarrow \Sigma$ can not answer more sophisticated questions on the text structure where the repeating element is not a single symbol, but rather, say, a substring. A very useful and famous example is the

Tandemly repeated DNA sequences are widespread throughout the human genome and show sufficient variability among individuals in a population that they have become important in several fields including genetic mapping, linkage analysis, and human identity testing. This importance has led to very active research in efficient algorithms to find tandem repeats (e.g. [11]). Tandem repeats may appear in various lengths. They are called *long tandem repeats* in the ranges of 100 kb to over 1 Mb, *short tandem repeats* in the ranges of 1 kb to 20 kb, and *microsatellites* or *simple sequence repeats* (SSRs) in the range of 1 to 6 base pairs. The repeat is not just a single repetition. In the case of STR’s, for example, the whole repetitive region may span up to 150 base pairs. Thus, an analysis of a repeat region may be a

query of the type: $\dots AAAAA$, which means: “find a region where a substring appears adjacent to itself five times”. Clearly one may think of more complex combinations, and interrelations between different substrings.

Other motivations for the problem may arise from poetry and music analysis, or author validation, among others. Various forms of poetry have quite a rigid order and organization. For example, the English or Shakespearean sonnet uses the rhyme scheme $ABABCDCDEFEGG$, the Italian sonnet using the rhyme scheme $ABBAABBACDECDE$. Various music forms also have a distinct structure. One may want to search for a repeated theme, for instance. Finally, in the \dots one is asked to verify authorship of an anonymous (or fraudulent) text (e.g. [16]). It is possible that certain writing structure will give the clue.

The above examples are a sample of diverse application areas encountering search problems that are not solved by state of the art methods in pattern matching. This need led us to introduce, in this paper, the \dots criterion, and to explore the efficiency of searching in this paradigm.

Definition 1. \dots Generalized Function Matching with Don’t Cares Problem (GFM_ϕ) \dots

$$\begin{aligned}
 & T \dots n \dots \Sigma_T \cup \{\phi\} \dots P \dots m \\
 & \Sigma_P \cup \{\phi\}, \dots \phi \dots \phi \dots \\
 & \dots \phi \dots \\
 & \dots i \dots f : \Sigma_P \rightarrow \Sigma_T^* \\
 & \dots i \dots f(P[0])f(P[1]) \dots \\
 & f(P[m-1])
 \end{aligned}$$

Function matching was a natural generalization of parameterized matching. However, relaxing the bijection restriction introduced non-trivial technical difficulties. Many powerful pattern matching techniques such as automata methods [17, 12], subword trees [23, 13], dueling [21, 4] and deterministic sampling [22] assume transitivity of the matching relation. For any pattern matching criteria where transitivity does not exist, the above methods do not help.

Examples of pattern matching with non-transitive matching relation are string matching with “don’t cares” [15], less-than matching [7], pattern matching with mismatches [1, 18] and swapped matching [19, 3, 6]. It is interesting to note that the efficient algorithms for solving the above problems all used convolutions as their main tool. Convolutions were introduced by Fischer and Paterson [15] as a technique for solving pattern matching problems where the match relation is not transitive. Indeed, in [2] convolutions played a key role in the efficient algorithm for function matching. It turns out that for generalized function matching, even convolutions are not powerful enough.

There are three main contributions in this paper.

1. The introduction of an important new type of generalized pattern matching, that of generalized function matching.
2. An example of an important new phenomenon in pattern matching. One where there is a significant complexity difference between the \dots

... and ... case. In all known pattern matching applications to date it was always the case that a polynomial time algorithm for infinite alphabet existed whenever a polynomial time algorithm for a bounded alphabet existed. A couple of examples are the exact matching case, where the time for both bounded and infinite alphabet is linear [17], pattern matching with mismatches, where the time for bounded alphabets is $O(n \log m)$ and for infinite alphabets is $O(n\sqrt{m \log m})$ [1]. To our knowledge, this is the first case where a pattern matching problem over a bounded alphabet can be solved in polynomial time but the infinite alphabet version is \mathcal{NP} -hard.

3. We show a polynomial time algorithm for the generalized function matching problem with don't cares over bounded alphabets. We prove that the generalized function matching problem with don't cares over infinite alphabets is \mathcal{NP} -hard.

2 The Bounded Alphabet Case

We reformulate the problem using simplified notations.

The ... is the following.

Text $T = t_0 t_1 \cdots t_{n-1}$, $t_i \in \Sigma_T \cup \{\phi\}$, $\forall i \in \{0, \dots, n-1\}$. Pattern $P = p_0 p_1 \cdots p_{m-1}$, $p_j \in \Sigma_P \cup \{\phi\}$, $\forall j \in \{0, \dots, m-1\}$.

All text locations i , for which $\exists f : \Sigma_P \rightarrow \Sigma_T^*$ such that $T(i) = f(P)$ (where we denote $T(i) = t_i t_{i+1} t_{i+2} \cdots t_{i+\ell}$, and $f(P) = f(p_0) f(p_1) \cdots f(p_{m-1})$).

Note that $T(i) = f(P)$ also in the case where there are ϕ 's in $T(i)$. Each such ϕ matches the symbol in $f(P)$ that is aligned with it. Notice also that the ϕ 's in the pattern can be replaced by different characters that are not in Σ_P . Formally, let t be the number of don't cares in the pattern. Define $\alpha_0, \alpha_1, \alpha_{t-1}$, ($\alpha_i \notin \Sigma_P$). Let P' be the string obtained by replacing the i th ϕ with α_i , $i = 0, \dots, t-1$.

Lemma 1. ... $P' \dots T \dots P$
 $\dots T$

Proof: It is clear that any generalized function matching of P' in T is also a generalized function matching of P in T . The other direction is also true since the only property that may be lost by the replacement is the injectivity. But since function matching (unlike parameterized matching) does not require this property, this is not a problem. \square

Note: The don't cares replacement is linear relatively to $|P|$, since the maximal number of don't cares in the pattern is $|P|$. Note also that we have made an implicit assumption that "don't care" in the pattern can match anything in the text. A more limiting assumption would be that a "don't care" can only match the image under f of a symbol from Σ_P . Here we choose to apply the broadest possible interpretation.

We will show in section 3 that the generalized function matching problem with don't cares in the ... only is \mathcal{NP} -hard for infinite alphabets. Lemma 1

guarantees that for infinite alphabets this problem is equivalent to the generalized function matching with don't cares problem. In this section we proceed to give a polynomial time algorithm for the generalized function matching problem with don't cares in the text only for bounded alphabets. The algorithm is a simple greedy algorithm.

Algorithm's Idea: For every text location i , consider every pattern symbol sequentially, trying to match it to all possible substrings starting where the last matched substring ended. Clearly this greedy strategy “blows up” for infinite alphabets, but for bounded alphabets, once all symbols were assigned to a substring there is only one possible assignment for every symbol.

We present a pseudocode of the algorithm and its time analysis.

```

Algorithm
1. For every text location  $i$  do:
    2. For each alphabet symbol  $\sigma \in \Sigma$  do:
        Construct all potential  $f_i(\sigma)$  as follows:
        For every  $i \leq k < \ell \leq n$ ,  $f_i(\sigma)$  is  $t_k \cdots t_\ell$ 
    endFor
    3. For every possible constructed value of  $f_i(p_0) \cdots f_i(p_{m-1})$ 
        construct  $f_i(P)$ .
    4. If  $T(i) = f_i(P)$  then accept and halt
endFor
5. { No acceptance for all  $i$  } reject
end Algorithm
    
```

Algorithm's Time: $O(n^{2|\Sigma_P|+1})$. It is easy to see that the algorithm can be streamlined and achieve time $O(n^{|\Sigma_P|+1})$. In any event, for a fixed bounded alphabet, the time is $O(n^c)$ for some c – clearly polynomial.

3 The Infinite Alphabet Case

Before proving the \mathcal{NP} -Completeness, it is necessary to convert the given optimization problem to a decision problem. It is possible to choose the most general way for this purpose, so that the only requirement for the decision problem is to answer the question “Is there a generalized function matching of the given pattern in the text?”. Formally, the generalized function matching decision problem is the following.

Definition 2. Generalized Function Matching with Don't Cares Decision Problem ($DGFM_\phi$)

$T = t_0 t_1 \cdots t_{n-1}$, $t_i \in \Sigma_T \cup \{\phi\}$, $\forall i \in \{0, \dots, n-1\}$
 $P = p_0 p_1 \cdots p_{m-1}$, $p_j \in \Sigma_P$, $\forall j \in \{0, \dots, m-1\}$
 Question: $\exists i, \exists f : \Sigma_P \rightarrow \Sigma_T^* \quad T(i) = f(P)$

Theorem 1. $DGFM_\phi \in NP$.

Proof: Clearly $DGFM_\phi \in NP$. Suppose there is an input $T = t_0t_1 \cdots t_{n-1}$, and $P = p_0p_1 \cdots p_{m-1}$. Guess text location i , and guess a function $f : \Sigma_P \rightarrow \Sigma_T^n$.

Now verify the following:

1. Construct $f(P)$ by replacing every p_j by $f(p_j)$.
2. Compare (character-by-character) $T(i)$ to $f(P)$.
3. If there is equality then accept, else reject.

Clearly the guess is polynomial in the input size: at most n elements for each of the (possibly) m different symbols of P , for a total of $O(nm)$. The verification is linear time.

We now show that $VC \leq_m^p DGFM_\phi$.

Definition 3. Vertex Cover(VC)

Given a graph $G = (V, E)$, k is a parameter. A vertex cover V' is a subset of vertices $V' \subseteq V$, $|V'| \leq k$ such that $\forall (v_{l_i}, v_{r_i}) \in E, v_{l_i} \in V' \vee v_{r_i} \in V'$.

The Reduction: Given a general input of the VC problem $G = (V, E)$, $V = (v_1v_2 \cdots v_n)$, $E = (e_1e_2 \cdots e_m)$, we construct the input for the generalized function matching.

Define:

$x_0, x_1, \dots, x_{k-1}, x_j \neq v_i : k$ different symbols that will be assigned generalized function matching. Refer to these symbols as pattern variables.

$p_0, p_1, \dots, p_{2m-1}, p_j \neq v_i : 2m$ different symbols whose task is pattern blocks borders.

$b_0, b_1, \dots, b_{2m-1}, b_j \neq v_i : 2m$ different symbols whose task is text blocks borders.

$\cdot_P, \cdot_T \neq x_i, p_i, b_i : \text{pattern and text block separators.}$

Denote by $\underbrace{\cdot_P \cdots \cdot_P}_t$ and $\underbrace{\cdot_T \cdots \cdot_T}_t$ means t sequential appearances of the pattern or text separator, respectively. We refer to them as sequential separator sets (of text or pattern).

$\underbrace{\phi \cdots \phi}_t : t$ sequential don't cares.

We are ready to construct the pattern.

$$P_R = \underbrace{\cdot_P \cdots \cdot_P}_t \frac{P}{Block(0)} \underbrace{\cdot_P \cdots \cdot_P}_t \frac{P}{Block(1)} \cdots \underbrace{\cdot_P \cdots \cdot_P}_t \frac{P}{Block(m-1)} \underbrace{\cdot_P \cdots \cdot_P}_t$$

where $\frac{P}{Block(i)} = p_{2i}x_0x_1 \cdots x_{k-1}p_{2i+1}$

The text is constructed as follows:

$$T_R = \underbrace{\cdot_T \cdots \cdot_T}_t \frac{T}{Block(0)} \underbrace{\cdot_T \cdots \cdot_T}_t \frac{T}{Block(1)} \cdots \underbrace{\cdot_T \cdots \cdot_T}_t \frac{T}{Block(m-1)} \underbrace{\cdot_T \cdots \cdot_T}_t$$

where $\frac{T}{Block(i)} = b_{2i} \underbrace{\phi \cdots \phi}_{k-1} v_{l_i} \underbrace{\phi \cdots \phi}_{k-1} v_{r_i} \underbrace{\phi \cdots \phi}_{k-1} b_{2i+1}$

v_{l_i} and v_{r_i} are the two vertices of edge e_i , and parameter t , the number of separators' repetitions will be specified later.

The idea of the reduction is to build a text block for every graph edge. This text block contains the two vertices that define the edge, a lot of don't cares, whose purpose will be explained in the following lemmas, and two more symbols, different from all others, that are located at the left and the right block borders. The number of pattern and text blocks is equal to the number of edges. Every two adjacent blocks are separated by block separators, (different in the text and pattern). The pattern blocks are almost identical (except for their borders), and can be matched to any text block.

Lemma 2.

Proof: Each pattern block contains k x_i 's and 2 different p_j 's. The pattern contains m pattern blocks, so without counting the pattern separators the size is: $|\frac{P}{Block(i)}|m = (k + 2)m$, which is $O(km)$.

In the text, every edge appears exactly once and for each of them there is one block. So, a text block contains 2 symbols for 2 edge vertices, 3 sets of $k - 1$ don't care and 2 borders symbols. There are $|E| = m$ text blocks, so not counting the text separators the size is: $(2 + 3k - 3 + 2)m = (3k + 1)m$, which is $O(mk)$.

Define t , the number of times sequential separators repeat, to be $(3k + 1)m$, for both pattern and text separators. The reduction stays polynomial size since the number of separators will be the main part of the pattern and text. The pattern size is therefore $(k + 2)m + (3k + 1)m(m + 1) = O(km^2)$, while the text size is also $O(km^2)$, it means both of them are polynomial in the input size. \square

Example: $V = (1, 2, \dots, 9)$, $E = ((1, 2), (2, 5), (3, 5), (1, 8))$, $k = 2$, $m = 4$, thus $t = 28$.

$$T_R = \underbrace{\dot{T} \dots \dot{T}}_{28} b_0 \phi_1 \phi_2 \phi b_1 \underbrace{\dot{T} \dots \dot{T}}_{28} b_2 \phi_2 \phi_5 \phi b_3 \underbrace{\dot{T} \dots \dot{T}}_{28} b_4 \phi_3 \phi_5 \phi b_5 \underbrace{\dot{T} \dots \dot{T}}_{28} b_6 \phi_1 \phi_8 \phi b_7 \underbrace{\dot{T} \dots \dot{T}}_{28}$$

$$P_R = \underbrace{\dot{P} \dots \dot{P}}_{28} p_0 x_0 x_1 p_1 \underbrace{\dot{P} \dots \dot{P}}_{28} p_2 x_0 x_1 p_3 \underbrace{\dot{P} \dots \dot{P}}_{28} p_4 x_0 x_1 p_5 \underbrace{\dot{P} \dots \dot{P}}_{28} p_6 x_0 x_1 p_7 \underbrace{\dot{P} \dots \dot{P}}_{28}$$

The next lemma is the main lemma required for the \mathcal{NP} -completeness proof.

Lemma 3. \exists $\dots \leq k \dots G = (V, E) \iff \dots i, \dots \exists f : \Sigma_P \rightarrow \Sigma_T^* \dots T(i) = f(P)$

Proof: \implies Assume \exists VC of size k in G . This means there is a set $\{v'_1, v'_2, \dots, v'_k\} = V' \subseteq V, \forall e_i, \exists v'_i \in e_i \cap V'$. In this case there is a match located at $i = 1$ and it is: $f(\dot{P}) = \dot{T}, f(x_i) = v_j, 0 \leq i, j \leq k - 1$, while for every $\frac{P}{Block(q)}$ match only one x_i to v_j , and the rest $x_r, 0 \leq r \leq k - 1, r \neq i$ will be set to don't cares that surround the v_j .

Lemma 4.

Proof: Based on the VC existence assumption, for every edge (text block) e_q , there is at least one symbol $v'_j \in V'$ which is in e_q . Thus, for every block q in

the text, there is at least one symbol $v_j \in V$ that belongs to the VC. So, choose this symbol for x_i . Since there are $k - 1$ don't cares on both sides of any v_j in the reduced text, set x_r , $0 \leq r \leq i - 1$ to a ϕ left of v_j and x_r , $i + 1 \leq r \leq k - 1$ to a ϕ right of v_j . Assign $\dot{:}P$ to $\dot{:}T$ since their number is the same. All other matches are between sets of separators and are not within those sets so there are no intersections by this matching. This way one can match any pattern blocks q to the parallel text block q , while the remaining symbols inside the text block will be set to p_{2q} and p_{2q+1} , $0 \leq q \leq 2m - 1$. Since to the left and right of v_j in any block q there are at least k symbols ($k - 1$ don't cares and one b_w), there is always at least one symbol for any p_r . A schematic of this function matching appears below.

$$\begin{array}{ccccccc} \underbrace{\dot{:}T \cdots \dot{:}T}_t & \xrightarrow{\text{Block}(0)} & \underbrace{\dot{:}T \cdots \dot{:}T}_t & \xrightarrow{\text{Block}(1)} & \underbrace{\dot{:}T \cdots \dot{:}T}_t & \cdots & \xrightarrow{\text{Block}(m-1)} & \underbrace{\dot{:}T \cdots \dot{:}T}_t \\ \hline \underbrace{\dot{:}P \cdots \dot{:}P}_t & \xrightarrow{\text{Block}(0)} & \underbrace{\dot{:}P \cdots \dot{:}P}_t & \xrightarrow{\text{Block}(1)} & \underbrace{\dot{:}P \cdots \dot{:}P}_t & \cdots & \xrightarrow{\text{Block}(m-1)} & \underbrace{\dot{:}P \cdots \dot{:}P}_t \end{array}$$

where the q blocks match is:

$$\begin{array}{ccccccc} b_{2q} \cdots \underbrace{\phi \cdots \phi}_{k-i-1} & \underbrace{\phi \cdots \phi}_i & v_j & \underbrace{\phi \cdots \phi}_{k-i-1} & \underbrace{\phi \cdots \phi}_i \cdots b_{2q+1} & & \\ \hline p_{2q} & x_0 x_1 \cdots x_{i-1} & x_i & x_{i+1} x_{i+2} \cdots x_{k-1} & p_{2q+1} & & \square \end{array}$$

\Leftarrow Assume \exists index i , function $f : \Sigma_P \rightarrow \Sigma_T^*$ such that $T(i) = f(P)$.

Lemma 5. $f(\dot{:}P) = \dot{:}T$

Proof: The number of repetitions of the pattern separator $\dot{:}P$ is $m(3k+1)(m+1)$. It can only be function matched to a symbol that appears at least $m(3k+1)(m+1)$ times. The only such text symbol is $\dot{:}T$ and it appears exactly $m(3k+1)(m+1)$ times. Therefore $f(\dot{:}P) = \dot{:}T$. □

Lemma 6. $\dots \dots \dots t \dots \dots \dots P \dots \dots \dots t \dots \dots \dots T$

Proof: Since $f(\dot{:}P) = \dot{:}T$, every $\dot{:}P$ is matched to either $\dot{:}T$ or ϕ . But pattern separators are located sequentially, in sets of $(3k + 1)m$, while the maximal number of sequential ϕ in the text is $k - 1$, which is less than the number of sequential separators. Thus, it is impossible to match one complete set to sequential don't cares only. On the other hand, the symbols immediately to the left and right of a text separator are b_r and b_l , respectively, and they are different from the text separator symbol. Thus, the only way for generalized function matching of a sequence of pattern separators is to match it to a sequence of text separators. □

The immediate conclusion from the above lemmas is that every set of sequential pattern separators is matched to a set of sequential text separators. Moreover, the beginning of this matching is from the first text character: $T(0) = f(P)$.

This means that for every i , the i th set of pattern separators is assigned to the i th set of text separators. This leads to the conclusion that the existence of a generalized function matching on a pattern and text constructed by the reduction causes every pattern block to be matched to the text block with the same index. Schematically we get:

$$\frac{b_l \underbrace{\phi \cdots \phi}_{k-1} v_l \underbrace{\phi \cdots \phi}_{k-1} v_r \underbrace{\phi \cdots \phi}_{k-1} b_r}{p_l x_0 x_1 \cdots x_{i-1} x_i x_{i+1} \cdots x_{k-1} p_r}$$

Lemma 7. $\forall p, \exists x_i, 0 \leq i \leq k - 1, f(x_i) = t_{q_p} t_{q_{p+1}} \cdots t_{q_r}, \exists j, t_{q_{p+j}} = v_l, t_{q_{p+j}} = v_r, \forall x_i, v_l, v_r$

Proof: We start by claiming that every pattern block has at least one x_i that is assigned to some text block symbols which are not solely don't cares. This is clear since the maximum number of sequential don't cares is $k - 1$, while the number of sequentially located x_i is k , at least one of them matches to non don't cares symbols.

Moreover, since p_l and p_r have to be matched to some not empty text symbols, b_l will be the first symbol of p_l 's matching, while b_r has to be the last symbol of p_r 's matching. Thus, for every pattern block q , $\exists x_i$, such that $f(x_i) = t_{q_p} t_{q_{p+1}} \cdots t_{q_r}$, where one of the t_{q_i} 's is either v_l or v_r , the two graph vertices that define the appropriate graph edge. \square

The idea for finding the vertex cover in case of generalized function matching is based on the next simple fact.

The Single Matching Claim:

$\forall x_i, \exists! (p_l, p_r)$

Proof: Based on the proven property that for every block there is at least one x_i that matches to a sequence that has v_l or v_r (or both) symbol, and adding the assumption that the size of all matchings is exactly 1, one can conclude that for every block there is exactly one x_i that is matched to some v_j , while the other pattern variables are matched to ϕ in this block. The reason is that around any v_j there are $k - 1$ ϕ , thus if x_i is matched to a v_j and the size of every matching is 1, the other pattern variables should be matched to ϕ in this particular block. This reasoning is true for all blocks. Thus, generalized function matching covers every text block and, as a result, every graph edge. The number of the pattern variables is k , so the vertex cover size is also $\leq k$. \square

The following lemma shows how to construct a generalized function matching that satisfies the requirement of the single matching claim, i.e. that $|f(x_i)| = 1$ for all pattern variables x_i . Once this is the case, the single matching claim guarantees the existence of a vertex cover of size k .

Lemma 8.

Let q be a pattern variable such that $|f(x_i)| > 1$ and $|f(x_w)| = 1, \forall w, 0 \leq w \leq k-1$.

Proof: By induction on q .

Base Case: $q = 1$. Assume there is exactly one x_i for which $|f(x_i)| > 1$, while for all other pattern variables $x_w, |f(x_w)| = 1$. It is necessary to show that in this case there is another generalized function matching such that $\forall x_w, 0 \leq w \leq k-1, w \neq j, |f(x_w)| = 1$.

Proof: Let x_i be the only pattern variable where $|f(x_i)| > 1$. We will show that there is a matching where x_0 is the only pattern variable where $|f(x_0)| > 1$. We do that iteratively by starting from the situation where x_i is the only pattern variable for which $|f(x_i)| > 1$, and creating a matching where x_{i-1} is the only pattern variable for which $|f(x_{i-1})| > 1$. This iteration is proven by the following lemma.

Lemma 9.

Let x_i be a pattern variable such that $|f(x_i)| = r > 1$ and $f(x_i) = t_{i_0}t_{i_0+1} \cdots t_{i_0+r-1}$. Let $f(x_{i-1}) = t_{i_0-1}$. Then $|f(x_{i-1})| > 1$ and $f(x_{i-1}) = t_{i_0-1}t_{i_0}t_{i_0+1} \cdots t_{i_0+r-2}$.

Proof: Suppose that the suggested change is not a correct generalized function matching. This means there is some text block p that is not the match of pattern block p using the suggested matching. The only pattern variables whose assignments were changed were x_i and x_{i+1} and they always follow each other. Therefore the generalized function matching of the pair $x_{i-1}x_i$ remains unchanged. Note that if there was injectivity before, it may now be lost, however, this is not a requirement in function matching. Thus in case of this local change the entire matching will stay correct. \square

The following lemma immediately follows, since one can iteratively “move to the left” the symbol x_i such that $|f(x_i)| > 1$ until it becomes x_0 .

Lemma 10.

Let x_0 be a pattern variable such that $|f(x_0)| > 1$ and $|f(x_w)| = 1, \forall w, 1 \leq w \leq k-1$.

Our current situation is that we have a generalized function matching where every pattern variable, except the first, is matched to a single text symbol. The following lemma shows how to construct a matching where every pattern variable is matched to a single text symbol, and then by the single matching claim we have a vertex cover.

Lemma 11. Let $f(x_0) = t_{i_0} t_{i_0+1} \cdots t_{i_0+r-1}$, $|f(x_0)| > 1$, $|f(x_i)| = 1, \forall i = 1, 2, \dots, k-1$, $\exists f'$ such that $|f'(x_i)| = 1, \forall i = 0, 1, \dots, k-1$

Proof: Let $f(x_0) = t_{i_0} t_{i_0+1} \cdots t_{i_0+r-1}$. Then construct matching f' such that $f'(x_i) = f(x_i), \forall i = 1, 2, \dots, k-1$, for every block j , $f'(b_{r_j}) = f(b_{r_j}), f'(b_{l_j}) = f(b_{l_j}) t_{i_0} t_{i_0+1} \cdots t_{i_0+r-2}$, and $f'(x_0) = t_{i_0+r-1}$.

f' is a generalized function matching since there is no problem with the b_{l_j} because they are different in every block. $f'(x_0)$ also must match in every block because $f(x_0)$ was a function and $f'(x_0)$ is simply the last element of $f(x_0)$. \square

Induction Step: Assume correctness for $q-1$. Prove the correctness for q .

Let x_l be the leftmost pattern variable that satisfies $|f(x_l)| > 1$. Then while all pattern variables x_i for $i < l$ are matched to exactly one text symbol, to the right of x_l there are $q-1 \neq 1$ pattern variables that are matched to more than one text symbol. Consider $f(x_{l+1})$. One can perform the same trick that had been done in the base case of the induction – construct a different generalized function matching by transferring all symbols of $f(x_l)$ to $f(x_{l+1})$, except the first symbol. The correctness of this operation is explained in a similar fashion to that of the base case of the induction. As a result, $|f(x_l)| = 1$, while $|f(x_{l+1})| > 1$ and this is another correct generalized function matching. If $|f(x_{l+1})|$ was equal to 1 in the original matching, continue performing this change in right direction until reaching the first x_j where x_{j+1} satisfies the condition: $|f(x_{j+1})| > 1$. Performing the operation now reduces the number of pattern variables that are matched to more than one symbol to $q-1$, so one can use the induction hypothesis and complete the proof. \square

4 Conclusion and Open Problems

We have shown what is, to our knowledge, the first known pattern matching problem that has a polynomial time solution for bounded alphabets but is \mathcal{NP} -hard for infinite alphabets. It would be interesting to find out whether the generalized function matching problem doesn't care has a polynomial time solution. Also, it is of interest to know whether generalized parameterized matching is \mathcal{NP} -complete or in \mathcal{P} . Finally, the generalized function matching problem has many applications. Thus it is important to find a good approximation to the problem.

References

1. K. Abrahamson. Generalized string matching. *SIAM J. Comp.*, 16(6):1039–1051, 1987.
2. A. Amir, A. Aumann, R. Cole, M. Lewenstein, and E. Porat. Function matching: Algorithms, applications, and a lower bound. In *Proc. 30th ICALP*, pages 929–942, 2003.
3. A. Amir, Y. Aumann, G. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. *Proc. 38th IEEE FOCS*, pages 144–153, 1997.

4. A. Amir, G. Benson, and M. Farach. An alphabet independent approach to two dimensional pattern matching. *SIAM J. Comp.*, 23(2):313–323, 1994.
5. A. Amir, K. W. Church, and E. Dar. Separable attributes: a technique for solving the submatrices character count problem. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 400–401, 2002.
6. A. Amir, R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. Overlap matching. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 279–288, 2001.
7. A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.
8. A. Apostolico and Z. Galil (editors). *Pattern Matching Algorithms*. Oxford University Press, 1997.
9. G.P. Babu, B.M. Mehtre, and M.S. Kankanhalli. Color indexing for efficient image retrieval. *Multimedia Tools and Applications*, 1(4):327–348, Nov. 1995.
10. B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th Annual ACM Symposium on the Theory of Computation*, pages 71–80, 1993.
11. G. Benson. Tandem repeats finder: a program to analyze dna sequence. *Nucleic Acids Research*, 27(2):573–580, 1999.
12. R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.
13. M. T. Chen and J. Seiferas. Efficient and elegant subword tree construction. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, chapter 12, pages 97–107. NATO ASI Series F: Computer and System Sciences, 1985.
14. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.
15. M.J. Fischer and M.S. Paterson. String matching and other products. *Complexity of Computation*, R.M. Karp (editor), *SIAM-AMS Proceedings*, 7:113–125, 1974.
16. D. Holmes. The evolution of stylometry in humanities scholarship. *Literary and Linguistic Computing*, 13(3):111–117, 1998.
17. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.
18. S. Rao Kosaraju. Efficient string matching. Manuscript, 1987.
19. S. Muthukrishnan. New results and open problems related to non-standard stringology. In *Proc. 6th Combinatorial Pattern Matching Conference*, pages 298–317. Lecture Notes in Computer Science 937, Springer-Verlag, 1995.
20. M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.
21. U. Vishkin. Optimal parallel pattern matching in strings. *Proc. 12th ICALP*, pages 91–113, 1985.
22. U. Vishkin. Deterministic sampling - a new technique for fast pattern matching. *SIAM J. Comp.*, 20:303–314, 1991.
23. P. Weiner. Linear pattern matching algorithm. *Proc. 14 IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

Approximate Distance Oracles for Graphs with Dense Clusters

Mattias Andersson¹, Joachim Gudmundsson², and Christos Levcopoulos¹

¹ Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden
`christos@cs.lth.se`, `mattias@cs.lth.se`

² Department of Mathematics and Computing Science, TU Eindhoven, 5600 MB, Eindhoven, the Netherlands
`h.j.gudmundsson@tue.nl`

Abstract. Let $\mathcal{H}_1 = (\mathcal{V}, \mathcal{F}_1)$ be a collection of N pairwise vertex disjoint $\mathcal{O}(1)$ -spanners where the weight of an edge is equal to the Euclidean distance between its endpoints. Let $\mathcal{H}_2 = (\mathcal{V}, \mathcal{F}_2)$ be a graph on \mathcal{V} with M edges of non-negative weight. The union of the two graphs is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{F}_1 \cup \mathcal{F}_2)$. We present a data structure of size $\mathcal{O}(M^2 + |\mathcal{V}| \log |\mathcal{V}|)$ that answers $(1 + \varepsilon)$ -approximate shortest path queries in \mathcal{G} in constant time, where $\varepsilon > 0$ is constant.

1 Introduction

The *shortest path* (SP) problem for weighted graphs with n vertices and m edges is a fundamental problem for which efficient solutions can now be found in any standard algorithms text, see also [8, 11, 18]. Lately the approximation version of this problem has also been studied extensively [2, 7, 9]. In numerous algorithms, the query version of the SP-problem frequently appears as a subroutine. In such a query, we are given two vertices and have to compute or approximate the shortest path between them. Thorup and Zwick [20] presented an algorithm for undirected weighted graphs that computes $(2k - 1)$ -approximate solutions to the query version of the SP problem in $\mathcal{O}(k)$ time, using a data structure that takes expected time $\mathcal{O}(kmn^{1/k})$ to construct and utilizes $\mathcal{O}(kn^{1+1/k})$ space. It is not an approximation scheme in the true sense because the value k needs to be a positive integer. Since the query time is essentially bounded by a constant, Thorup and Zwick refer to their queries as approximate *shortest path queries*.

We focus on the geometric version of this problem. A geometric graph has vertices corresponding to points in \mathbb{R}^d and edge weights from a Euclidean metric. Throughout this paper we will assume that d is a constant. A geometric graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is said to be a t -spanner for \mathcal{V} , if for any two points p and q in \mathcal{V} , there exists a path of length at most t times the Euclidean distance between p and q . For geometric graphs, also, considerable previous work exists on the shortest path and related problems. A good survey can be found in [17], see also [3, 6, 10, 19]. The geometric query version was recently studied by Gudmundsson et al. [13, 14]

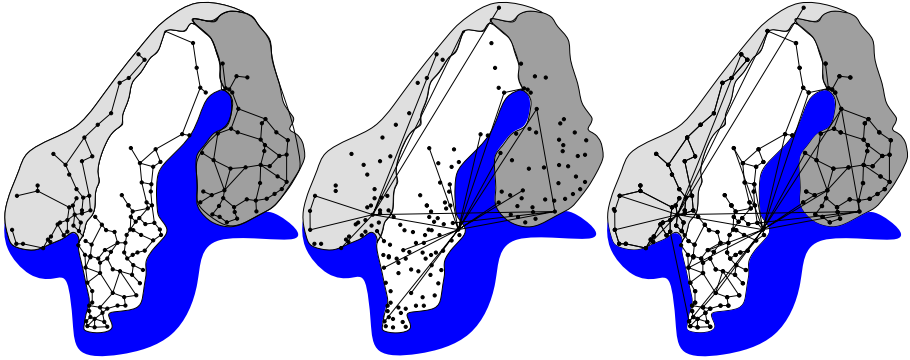


Fig. 1. The three figures models parts of the transportation network in Norway, Sweden and Finland. (a) The domestic railway network in the three countries (not complete). (b) The railway connections between the countries together with the main air and sea connections within, and between, Norway, Sweden and Finland. (c) The two networks combined into one graph \mathcal{G}

and they presented the first data structure that answers approximate shortest-path queries in constant time, provided that the input graph is a t -spanner for some known constant $t > 1$. Their data structure uses $\mathcal{O}(|\mathcal{V}| \log |\mathcal{V}|)$ space and can be constructed in time $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$.

In this paper we extend the results in [13, 14] to hold also for “islands” of t -spanners, i.e., a set of N vertex disjoint t -spanners inter-connected through “airports” i.e. M edges of arbitrary non-negative weight. We construct a data structure that can answer $(1 + \varepsilon)$ -approximate shortest path queries in constant time. The data structure uses $\mathcal{O}(M^2 + |\mathcal{V}| \log |\mathcal{V}|)$ space and can be constructed in time $\mathcal{O}(|\mathcal{E}| + (M^2 + |\mathcal{V}|) \log |\mathcal{E}|)$. Hence, for $M = \mathcal{O}(\sqrt{|\mathcal{E}|})$ the bound is essentially the same as in [13, 14].

We claim that the generalization studied is natural in many applications. Consider for example the freight costs within Norway, Sweden and Finland, see Fig. 1. The railway network and the road network within a country are usually t -spanners for some small value t , and the weight (transport cost) of an edge is linearly dependent on the Euclidean distance. In Fig. 1a the railway networks of Norway, Sweden and Finland (although not complete) is shown. The weight of an edge is dependent on the Euclidean distance between its endpoints. Hence, each country’s railway network and road network can most often be modeled as a Euclidean t -spanner for some small constant t . (Places that are not reachable by train are treated as single t -spanners containing only one point, for example Haugesund on the west coast of Norway is only reachable by boat.). Apart from these edges there are also edges that model, for example, air freight, sea freight, or inter-connecting railway transports. An example of this is shown Fig. 1b, where the main air and sea routes together with the inter-connecting railway tracks are shown. The weight of these edges can be completely independent of the Euclidean distance, as is usually the case when it comes to air fares. The

reason why inter-connecting railway transport is included in the latter set of edges is because the railway networks of different countries are usually sparsely connected. For example, there are one connection between Sweden and Finland, four between Norway and Sweden, and zero between Finland and Norway.

In [14] it was shown that an approximate shortest-path distance oracle can be applied to a large number of problems, for example, finding a shortest obstacle-avoiding path between two vertices in a planar polygonal domain with obstacles and interesting query versions of closest pair problems. The extension presented in this paper also generalizes the results for the above mentioned problems.

The main idea for obtaining our results is to develop a method to efficiently combine existing methods for $\mathcal{O}(1)$ -spanners with methods for general graphs. One problem, for example, may be given a starting point p and a destination q , which should be the first airport to travel to, since (in theory) there might be a non-constant number of airports on p 's island? In order to achieve this, we determine a small number, $\mathcal{O}(M)$, of representative ‘‘junction’’ points, so that every point p in the graph is represented by exactly one such junction point $r(p)$, located on the same island as p . All endpoints of the M edges of \mathcal{H}_2 are also treated as such junction points. For all pairs of junction points we precompute approximate distances, using space $\mathcal{O}(M^2)$. For any two points p and q , an approximately shortest path between them is found either by only using edges of one of the $\mathcal{O}(1)$ -spanners, or by following a path from p to its representative junction point $r(p)$, then from $r(p)$ to $r(q)$, and finally from $r(q)$ to q . In order to choose such a small set of suitable representative junction points we present, in Section 2.4, a partition of space which may be useful also in other applications. In Section 3 we show general correctness, and in Section 4 we mention some refinements and extensions of the main results. Due to lack of space, many proofs are omitted in this extended abstract.

Our model of computation is the traditional algebraic computation model with the added power of indirect addressing. We will use the following notation. For points p and q in \mathcal{R}^d , $|pq|$ denotes the Euclidean distance between p and q . If \mathcal{G} is a geometric graph, then $\delta_{\mathcal{G}}(p, q)$ denotes the Euclidean length of a shortest path in \mathcal{G} between p and q . If P is a path in \mathcal{G} between p and q of length Δ with $\delta_{\mathcal{G}}(p, q) \leq \Delta \leq (1 + \epsilon) \cdot \delta_{\mathcal{G}}(p, q)$, then P is a $(1 + \epsilon)$ -approximate shortest path for p and q . The main result of this paper is stated in the following theorem:

Theorem 1. Let $\mathcal{H}_1 = (\mathcal{V}, \mathcal{F}_1)$ and $\mathcal{H}_2 = (\mathcal{V}, \mathcal{F}_2)$ be two $\mathcal{O}(1)$ -spanners of N vertices \mathcal{V} with t edges per vertex ($t > 1$). Let \mathcal{H}_2 have M edges. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E} = \{\mathcal{F}_1 \cup \mathcal{F}_2\})$ be the union of \mathcal{H}_1 and \mathcal{H}_2 . Then there exists a $(1 + \epsilon)$ -approximate shortest path oracle for \mathcal{G} with space $\mathcal{O}((|\mathcal{E}| + M^2) \log |\mathcal{V}|)$ and query time $\mathcal{O}(M^2 + |\mathcal{V}| \log |\mathcal{V}|)$ for any $0 < \epsilon < 1$.

The set of pairwise vertex disjoint t -spanners of \mathcal{H}_1 is called the ‘‘islands’’ of \mathcal{G} and will be denoted $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \dots, \mathcal{G}_N = (\mathcal{V}_N, \mathcal{E}_N)$. An edge $(u, v) \in \mathcal{F}_2$ is

said to be an s -separated edge (even though both its endpoints may belong to the same island). A vertex $v \in \mathcal{V}_i$ incident to an edge in \mathcal{H}_2 is called an s -airport, for simplicity (even though these vertices may represent any kind of junction point). The set of all airports of \mathcal{V}_i is denoted \mathcal{C}_i . Note that the total number of airports is $\mathcal{O}(M)$ since the number of inter-connecting edges is M .

2 Tools

In the construction of the distance oracle we will need several tools, among them the well-separated pair decomposition by Callahan and Kosaraju [5], a graph pruning tool by Gudmundsson et al. [14, 15] and well-separated clusters by Krznicaric and Levkopoulos [16]. In this section we briefly recollect these tools. In section 2.4 we also show a useful tool that clusters points with respect to a subset of representative points, as described in the introduction.

2.1 Well-Separated Pair Decomposition

Definition 1. Let S be a set of points in \mathcal{R}^d and $s > 0$. A well-separated pair decomposition (WSPD) of S is a collection of pairs $\{A, B\}$ of subsets of S such that

$$|xx'| \leq (2/s)|x'y'| \text{ and } |x'y'| \leq (1 + 4/s)|xy|$$

Lemma 1. Let A and B be two disjoint sets of points in \mathcal{R}^d . Then there exists a WSPD pair $\{A, B\}$ such that

$$|xx'| \leq (2/s)|x'y'| \text{ and } |x'y'| \leq (1 + 4/s)|xy|$$

Definition 2 ([5]). Let S be a set of points in \mathcal{R}^d and $s > 0$. A well-separated pair decomposition (WSPD) of S is a collection of pairs $\{A_i, B_i\}$, $1 \leq i \leq m$, such that

- (1) $A_i \cap B_i = \emptyset$, $i = 1, \dots, m$,
- (2) $\{A_i, B_i\}$ is a WSPD pair, $i = 1, \dots, m$.

The integer m is called the size of the WSPD. Callahan and Kosaraju show how such a WSPD can be computed. They start by constructing in $\mathcal{O}(n \log n)$ time, a split tree T having the points in S as leaves. Given this tree, they show how a WSPD of size $m = \mathcal{O}(s^d n)$ can be computed in time $\mathcal{O}(s^d n)$. In this WSPD, each pair $\{A_i, B_i\}$ is represented by two nodes u_i and v_i of T . That is, A_i and B_i are the sets of all points stored at the leaves of the subtrees rooted at u_i and v_i , respectively.

Theorem 2. Let S be a set of points in \mathcal{R}^d and $s > 0$. Then there exists a WSPD of S of size $m = \mathcal{O}(s^d n)$ that can be computed in time $\mathcal{O}(n \log n + s^d n)$.

2.2 Pruning a t -Spanner

In [14] it was shown how the WSPD can be used to prune an existing t -spanner. Assume that we are given a t -spanner $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Compute a WSPD $\{A_i, B_i\}$, $1 \leq i \leq \ell$, for \mathcal{V} , with separation constant $s = 4(1 + (1 + \epsilon)t)/\epsilon$ and $\ell = O(n)$. Let $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ be the graph that contains for each i , exactly one (arbitrary) edge (x_i, y_i) of E with $x_i \in A_i$ and $y_i \in B_i$, provided such an edge exists. It holds that \mathcal{G}' is a $(1 + \epsilon)$ spanner of \mathcal{G} , and hence:

Fact 1. $(\mathcal{G}, \mathcal{G}') \text{ is a } (1 + \epsilon) \text{ spanner of } \mathcal{G} \text{ and } \epsilon > 0 \implies t \geq 1$
 $\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{G}' = (\mathcal{V}, \mathcal{E}'), t > 1, n \text{ points in } \mathcal{V}, m \text{ edges in } \mathcal{G}$
 $(1 + \epsilon) \mathcal{G}' \text{ is a spanner of } \mathcal{G} \implies \mathcal{O}(n) \text{ edges in } \mathcal{G}', \mathcal{O}(m + n \log n) \text{ time}$

2.3 Well-Separated Clusters

Let \mathcal{S} be a set of points in the plane, and let $b \geq 1$ be a real constant. Let the b -spanner of \mathcal{R} , abbreviated $\text{rd}(\mathcal{R})$, be the diameter of the smallest axis-aligned rectangle containing \mathcal{R} . We may now consider the following cluster definitions from [16]:

Definition 3. $\mathcal{D} \subseteq \mathcal{S}$ is a b -cluster if $\mathcal{D} \subseteq \mathcal{S}$ and $\text{rd}(\mathcal{D}) \leq b \cdot \text{rd}(\mathcal{S})$

Definition 4. A hierarchy of b -clusters \mathcal{S} is a sequence of sets $\mathcal{S} = \mathcal{S}_0 \supseteq \mathcal{S}_1 \supseteq \dots \supseteq \mathcal{S}_k$ such that $\mathcal{S}_0 = \mathcal{S}$ and for each $i < k$, \mathcal{S}_i is a b -cluster of \mathcal{S}_{i+1} . A b -cluster \mathcal{C} is a b -cluster of \mathcal{A} if $\mathcal{C} \subseteq \mathcal{A}$ and $\mathcal{C} \subseteq \mathcal{B} \subseteq \mathcal{A}$ for some b -cluster \mathcal{B} of \mathcal{A} .

The following observation is straightforward.

Observation 1. $A, B \text{ are } b \text{-clusters of } \mathcal{S} \implies |x'y'| \leq (1 + 2/b)|xy|$
 $A \text{ contains } x, y \text{ and } B \text{ contains } x', y'$

Assume w.l.o.g that $|x'y'| \geq |xy|$ and that $\text{rd}(A) \geq \text{rd}(B)$. This means

$$|x'y'| \leq |xy| + 2\text{rd}(A) \leq |xy| + 2|xy|/b \implies |x'y'| \leq (1 + 2/b)|xy|$$

The cluster tree can also be computed efficiently.

Theorem 3. \mathcal{S} is a set of n points in \mathbb{R}^d and $b \geq 1$. The cluster tree of \mathcal{S} can be computed in $\mathcal{O}(n \log n)$ time.

2.4 Partitioning Space into Small Cells

Given a set \mathcal{V} of n points in \mathbb{R}^d and a subset $\mathcal{V}' \subseteq \mathcal{V}$, we will show how to associate a representative point $r \in \mathcal{V}$ with each point $p \in \mathcal{V}$, such that the distance $|pr| + |rq|$, for any point $q \in \mathcal{V}'$, is a good approximation of the distance $|pq|$. The total number of representative points is $\mathcal{O}(|\mathcal{V}'|)$. The idea is to partition space into cells, such that all points included in a cell may share a common representative point.

We will use the following fact by Mount et al. [4]:

Fact 2. (.....) S n \mathbb{R}^d
..... $c_{d,\varepsilon} \leq d \lceil 1 + 6d/\varepsilon \rceil^d$ $\mathcal{O}(dn \log n)$
..... $\mathcal{O}(dn)$
..... $\varepsilon > 0$ $q \in \mathbb{R}^d$ $(1 + \varepsilon)$ q
..... S $\mathcal{O}(c_{d,\varepsilon} \log n)$
..... $\varepsilon_N > 0, q \in \mathbb{R}^d$ $k, 1 \leq k \leq n$ k
..... $(1 + \varepsilon)$ $\mathcal{O}((c_{d,\varepsilon} + kd) \log n)$

Computing Representative Points. As input we are given the point set \mathcal{V} , $\mathcal{V}' \subseteq \mathcal{V}$ and a positive real value $\tau_1 < 1$. Set $\varepsilon = (1 + \sqrt{2})\tau_1/\sqrt{8}$. As a pre-processing step we compute the b -cluster tree \mathcal{T} of \mathcal{V}' with $b = 10/\varepsilon^2$, as described in Theorem 3.

For a level i in \mathcal{T} let $\nu(\mathcal{D}_1), \dots, \nu(\mathcal{D}_{\ell_i})$ be the nodes at that level, where $\mathcal{D}_1, \dots, \mathcal{D}_{\ell_i}$ are the associated clusters. For each cluster \mathcal{D}_j pick an arbitrary vertex d_j as the center point of \mathcal{D}_j . The set of the ℓ_i center points is denoted $\mathcal{D}(i)$. Perform the following four steps for each level i of \mathcal{T} .

1. Compute an approximate nearest neighbor structure with $\mathcal{D}(i)$ as input, as described in Fact 2.
2. For each center point d_j in $\mathcal{D}(i)$ compute the $(1 + \varepsilon)$ -approximate nearest neighbor of d_j . The point returned by the structure is denoted v_j , where $v_j \neq d_j$.
3. For each cluster \mathcal{D}_j construct two squares; $is(\mathcal{D}_j)$ and $os(\mathcal{D}_j)$ with centers at d_j and side length $2\alpha = 2(1 + 1/\varepsilon) \cdot rd(\mathcal{D}_j)$ and $2\beta = \frac{2\varepsilon|d_j, v_j|}{(1 + \varepsilon)(1 + 2/b)}$ respectively, where $\alpha < \beta$. The two squares are called the inner and outer shells of \mathcal{D}_j , and the set theoretical difference between the inner and the outer shell is denoted the of \mathcal{D}_j .
4. The inner shell of \mathcal{D}_j is recursively partitioned into four equally sized squares, until each square s either
 - (a) is completely included in the union of the outer shells of the children of $\nu(\mathcal{D}_j)$. In this case the square is deleted and, hence, not further partitioned. Or,
 - (b) has diameter at most $\frac{\varepsilon}{1 + \varepsilon} \cdot K$, where K is the smallest distance between a point within s and a point in \mathcal{D}_j . A $(1 + \varepsilon)$ -approximation of K can be computed in time $\mathcal{O}(\log |\mathcal{D}_j|)$. This implies that the diameter of s is bounded by $\varepsilon \cdot K$.

The resulting cells are denoted Note that, due to step 4a, every inner cell is empty of points from \mathcal{D}_j .

Finally, after all levels of \mathcal{T} have been processed, we assign a representative point to each point p in \mathcal{V} . Preprocess all the produced cells and perform a point-location query for each point. If p belongs to a doughnut cell then the center point of the associated cluster (see step 1) is the representative point of

p . Otherwise, if p belongs to an inner cell C and p is the first point within C processed in this step then $rep(C)$ is set to p . If p is not the first point then $rep(p) = rep(C)$. Further, note that an inner cell may overlap with the union of the outer shells of the children of $v(D_j)$. If a point is included in both an inner cell and an outer shell, we treat it as if it belonged to the inner cell, and assign a representative point as above.

We state the main result of this section.

Theorem 4. Let $\mathcal{V} \subseteq \mathbb{R}^d$ be a set of n points in \mathbb{R}^d , $\mathcal{V}' \subseteq \mathcal{V}$ be a subset of \mathcal{V} with $|\mathcal{V}'| \leq n$, and $\tau_1 < 1$. For each point $p \in \mathcal{V}$, let $r(p) \in \mathcal{V}$ be a representative point and $h \in \mathcal{V}'$ be a point in \mathcal{V}' . Then, for any point $p \in \mathcal{V}$, we have

$$\min\{|p, r(p)|, |r(p), h|\} \leq \tau_1 |p, h|.$$

The time to compute $r(p)$ and h for all $p \in \mathcal{V}$ is $\mathcal{O}(|\mathcal{V}'| n)$.

3 Constructing the Oracle

This section is divided into three subsections: first we present the construction of the structure, then how queries are answered and, finally the analysis is presented.

Consider two graphs $\mathcal{H}_1 = (\mathcal{V}, \mathcal{F}_1)$ and $\mathcal{H}_2 = (\mathcal{V}, \mathcal{F}_2)$ with the same vertex set, where \mathcal{H}_1 is a collection of N vertex disjoint Euclidean t -spanners $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, $1 \leq i \leq N$, with m edges where $t > 1$ is a constant, and \mathcal{H}_2 is a graph with M edges of non-negative weight. The union of the two graphs is denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E} = \{\mathcal{F}_1 \cup \mathcal{F}_2\})$.

3.1 Constructing the Basic Structures

In this section we show how to pre-process \mathcal{G} in time $\mathcal{O}(m + (M^2 + n) \log n)$ such that we obtain three structures that will help us answer approximate distance queries in constant time. We will assume that the number of edges in each subgraph is linear with respect to the number of vertices in \mathcal{V}_i , if not the subgraph is pruned using Fact 1. Hence, we can from now on assume that $\#\mathcal{E}_i = \mathcal{O}(|\mathcal{V}_i|)$.

Let \mathcal{V}' be the set of vertices in \mathcal{V} incident on an inter-connecting edge. Now we can apply Theorem 4 with parameters \mathcal{V} , $\mathcal{V}' = \Gamma'$ and τ_1 to obtain a representative point for each point in \mathcal{V} .

Now we are ready to present the three structures:

Oracle A: An oracle that given points p and q returns a 3-tuple $[SI, r(p), r(q)]$, where SI is a boolean with value ‘true’ if p and q belongs to the same island, otherwise it is ‘false’, and $r(p)$ and $r(q)$ are the representative points for p and q respectively.

Oracle B: An approximate distance oracle for any pair of points belonging to the same island.

Matrix D: An $\mathcal{O}(M) \times \mathcal{O}(M)$ matrix. For each pair of representative points, p and q , D contains the approximate shortest distance between p and q .

The representative point of a point p is denoted $r(p)$, and the set of all representative points of \mathcal{V}_i and \mathcal{V} is denoted Γ_i and Γ , respectively. Note that $\mathcal{C}_i \subseteq \Gamma_i$. Now we turn our attention to the construction of the oracles and the matrix.

Oracle A: The oracle is a 4-level tree, denoted \mathcal{T} , with the points of \mathcal{V} corresponding to the leaves of \mathcal{T} . The parents of the leaves correspond to the representative points of \mathcal{V} and their parents correspond to the islands $\mathcal{G}_1, \dots, \mathcal{G}_N$ of \mathcal{G} . Finally, the root of \mathcal{T} corresponds to \mathcal{G} . Since the representative points already are computed, the tree \mathcal{T} can be constructed in linear time. The root is at level 0 and the leaves are at level 3 in \mathcal{T} .

Assume that one is given two points p and q . Follow the paths from p and q respectively to the root of \mathcal{T} . If p and q have the same ancestor at level 1 then they lie on the same island and hence SI is set to ‘true’, otherwise to ‘false’. Finally, the ancestor of p and the ancestor of q at level 2 correspond to the representative points of p and q . Obviously a query can be answered in constant time since the number of levels in \mathcal{T} is four.

Oracle B: This oracle is the structure that is easiest to build since we can apply the following result to each of the islands (see also [14]).

Fact 3. Let \mathcal{V} be a set of n points in R^d with τ_2 -separation. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with t edges, where $t > 1$, and m vertices. Then \mathcal{G} can be processed in $\mathcal{O}(n \log n)$ time to produce a data structure that can answer shortest path queries between points $p, q \in \mathcal{V}$ in $\mathcal{O}(n \log n)$ time. The total time is $\mathcal{O}(n \log n)$ and the space is $\mathcal{O}(n \log n)$.

Hence, oracle B will actually be a collection of oracles, one for each island. Given two points p and q the appropriate oracle can easily be found in constant time using a similar construction as for oracle A . Thus, after $\mathcal{O}(n \log n)$ preprocessing using $\mathcal{O}(n \log n)$ space, $(1 + \tau_2)$ -approximate shortest path queries between points on the same island can be answered in constant time.

Matrix D: For each i , $1 \leq i \leq N$, compute the WSPD of Γ_i with separation constant $s = (\frac{1+\tau_2+\tau_3}{\tau_3-\tau_2})$. As output we obtain a set of well-separated pairs $\{A_i, B_i\}_{1 \leq i \leq w_i}$, such that $w_i = \mathcal{O}(\#\mathcal{C}_i)$. Next, construct the non-Euclidean graph $\mathcal{F} = (\Gamma, \{\mathcal{E}' \cup \mathcal{F}_2\})$, where \mathcal{E}' is constructed as follows. For each Γ_i and each well-separated pair $\{A_j, B_j\}$ of the WSPD of Γ_i select two (arbitrary) representative points $a_j \in A_j$ and $b_j \in B_j$. Add the edge (a_j, b_j) to \mathcal{E}' with weight $B_i(a_j, b_j)$, where $B_i(p, q)$ denotes a call to oracle B_i for \mathcal{G}_i with parameters p and q . Note that the graph \mathcal{F} will have $\mathcal{O}(M)$ vertices and edges.

Let D be an $|\Gamma| \times |\Gamma|$ matrix. For each representative point $p \in \Gamma$ compute the single-source shortest path in \mathcal{F} to every point $q \in \Gamma$ and store the distance of each path in $D[p, q]$. The total time for this step is $\mathcal{O}(M^2 \log M)$, and it can be obtained by running Dijkstra’s algorithm M times.

Lemma 2. Let A be a matrix with M rows and M columns. Let B be a matrix with M rows and M columns. Let D be a matrix with M rows and M columns. Then A, B, D can be processed in $\mathcal{O}(m + (M^2 + n) \log n)$ time to produce a data structure that can answer shortest path queries between points $p, q \in \Gamma$ in $\mathcal{O}(M^2 + n \log n)$ time.

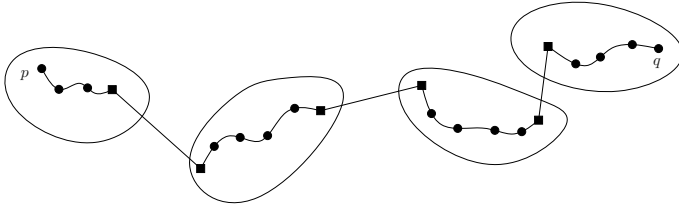


Fig. 2. Illustrating the approximate shortest path between p and q . The boxes illustrate the “airports” along the path

The lemma is obtained by adding up the complexity for the pre-processing together with the cost of building each structure. Recall that as pre-processing steps we first pruned the subgraphs and then we computed the representative point for each point in \mathcal{V} . This was done in $\mathcal{O}(m + n \log n)$ time using $\mathcal{O}(n \log n)$ space, according to Fact 1 and Theorem 4. Next, oracle A was constructed in linear time using linear space, followed by the construction of oracle B which, according to Fact 3 was done in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(n \log n)$ space. Finally, the matrix D was constructed by first computing the graph \mathcal{F} . Then a single-source shortest path query was performed for each vertex in \mathcal{F} . Since the complexity of \mathcal{F} is $\mathcal{O}(M)$ it follows that D was computed in time $\mathcal{O}(M^2 \log n)$ using $\mathcal{O}(M^2)$ space. Hence, adding up these bounds gives the lemma.

3.2 Querying

Given the two oracles and the matrices presented above the query algorithm is very simple, see pseudo-code below. Let $r(p)$ denote the representative point of $p \in \mathcal{V}$. Now assume that we are given two points p and q . If p and q do not belong to the same island we return the sum of $B(p, r(p))$, $D(r(p), r(q))$ and $B(r(q), q)$. If p and q belong to the same island then we query Oracle B with input p, q and return the value obtained from the oracle. Obviously this is done in constant time.

QUERY(p, q)

1. $[\text{SameIsland}, r(p), r(q)] \leftarrow A(p, q)$
2. $distance \leftarrow B(p, r(p)) + D(r(p), r(q)) + B(r(q), q)$
3. **if** SameIsland **then**
4. $distance \leftarrow \min(distance, B(p, q))$
5. return $distance$

3.3 Correctness

Let $\delta_{\mathcal{G}}(p, q)$ be a shortest path in a graph \mathcal{G} between two points p and q . Set τ_2, τ_3 and τ_5 to be positive real values such that $1 + \varepsilon = (1 + \tau_2)(1 + \tau_3)(1 + \tau_5)$.

Observation 2. $\dots p \dots q \dots \mathcal{V}_i \dots B(p, q) \leq (1 + \tau_2) \cdot \delta_{\mathcal{G}_i}(p, q)$

Observation 3. $\dots p \in \mathcal{V}_i \dots \delta_{\mathcal{G}_i}(p, r(p)) \leq (1 + \tau_4) \cdot \delta_{\mathcal{G}}(p, r(p))$. $\tau_4 = t \cdot \tau_1$

Lemma 3. $\dots p \dots q \dots \mathcal{V}_i \dots D(p, q) \leq (1 + \tau_3) \cdot \delta_{\mathcal{G}_i}(p, q)$

Note that it suffices to prove that $D(p, q) \leq \frac{1+\tau_3}{1+\tau_2} \cdot B(p, q)$, according to Observation 2. The proof is done by induction on the Euclidean length of (p, q) .

Recall that $\mathcal{F} = (\Gamma, \mathcal{E}')$ was constructed to build the matrix D . Assume that (p, q) is the closest pair of Γ . In this case there exists a well-separated pair $\{A_j, B_j\}$ such that $A_j = \{p\}$ and $B_j = \{q\}$ otherwise (p, q) could not be the closest pair. Hence the claim holds since there is an edge in \mathcal{F} of weight $B(a_j, b_j)$.

Assume that the lemma holds for all pairs in Γ closer than $|pq|$ to each other.

If $(p, q) \notin \mathcal{F}$ then there exists an edge (x, y) in \mathcal{F} and a well-separated pair $\{A_j, B_j\}$ such that $x, p \in A_j$ and $y, q \in B_j$. According to the induction hypothesis there is path between p and x of weight $\frac{1+\tau_3}{1+\tau_2} \cdot D(p, q)$ and a path between y and q of weight $\frac{1+\tau_3}{1+\tau_2} \cdot D(p, q)$. Also, the weight of the edge (x, y) is $B(x, y)$. Putting together the weights we obtain that

$$\begin{aligned} \delta_{\mathcal{F}}(p, q) &\leq \frac{1 + \tau_3}{1 + \tau_2} B(p, x) + B(x, y) + \frac{1 + \tau_3}{1 + \tau_2} B(y, q) \\ &\leq \left(2 \frac{1 + \tau_3}{1 + \tau_2} (2/s) + (1 + 4/s) \right) \cdot B(p, q) \\ &= \frac{1 + \tau_3}{1 + \tau_2} B(p, q) = (1 + \tau_3) \cdot \delta_{\mathcal{G}_i}(p, q) \end{aligned}$$

In the third step we used the fact that $s = \frac{1+\tau_2+\tau_3}{\tau_3-\tau_2}$.

Corollary 1. $\dots p \dots q \dots \mathcal{V} \dots D(p, q) \leq (1 + \tau_3) \cdot \delta_{\mathcal{G}}(p, q)$

Lemma 4. $\dots p, q \in \mathcal{V} \dots$

$$\delta_{\mathcal{G}}(p, r(p)) + \delta_{\mathcal{G}}(r(p), r(q)) + \delta_{\mathcal{G}}(r(q), q) \leq (1 + \tau_5) \cdot \delta_{\mathcal{G}}(p, q).$$

Lemma 5. $\dots p \dots q \dots \mathcal{V} \dots$

$$\delta_{\mathcal{G}}(p, q) \leq B(p, r(p)) + D(r(p), r(q)) + B(r(q), q) \leq (1 + \varepsilon) \cdot \delta_{\mathcal{G}}(p, q).$$

$$\begin{aligned}
 \delta_{\mathcal{G}}(p, q) &\leq B(p, r(p)) + D(r(p), r(q)) + B(r(q), q) \\
 &\leq (1 + \tau_2) \cdot \delta_{\mathcal{G}_i}(p, r(p)) + (1 + \tau_3) \cdot \delta_{\mathcal{G}}(r(p), r(q)) \\
 &\quad + (1 + \tau_2) \cdot \delta_{\mathcal{G}_j}(r(q), q) \\
 &\leq (1 + \tau_2)(1 + \tau_4) \cdot \delta_{\mathcal{G}}(p, r(p)) + (1 + \tau_3) \cdot \delta_{\mathcal{G}}(r(p), r(q)) \\
 &\quad + (1 + \tau_2)(1 + \tau_4) \cdot \delta_{\mathcal{G}}(r(q), q) \\
 &< (1 + \tau_4)(1 + \tau_3) \cdot (\delta_{\mathcal{G}}(p, r(p)) + \delta_{\mathcal{G}}(r(p), r(q)) + \delta_{\mathcal{G}}(r(q), q)) \\
 &\leq (1 + \tau_4)(1 + \tau_3)(1 + \tau_5) \cdot \delta_{\mathcal{G}}(p, q) \\
 &= (1 + \epsilon) \cdot \delta_{\mathcal{G}}(p, q)
 \end{aligned}$$

On line 1 we used Observation 2 together with Lemma 3. On the following line we used Observation 3, applied Lemma 4 and finally replaced $(1 + \tau_2)(1 + \tau_3)(1 + \tau_5)$ with $(1 + \epsilon)$.

Putting together Lemma 2 and Lemma 5 gives us Theorem 1. For convenience we restate Theorem 1 below:

Theorem 1. Let $\mathcal{H}_1 = (\mathcal{V}, \mathcal{F}_1)$ and $\mathcal{H}_2 = (\mathcal{V}, \mathcal{F}_2)$ be two vertex-disjoint Euclidean t -spanners ($t > 1$) on a set of n vertices. Let M and N be positive integers. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E} = \{\mathcal{F}_1 \cup \mathcal{F}_2\})$ be the graph formed by the union of \mathcal{H}_1 and \mathcal{H}_2 . Then there exists a data structure \mathcal{D} that stores \mathcal{G} and answers distance queries in constant time. The space used by \mathcal{D} is $\mathcal{O}((|\mathcal{E}| + M^2) \log n)$. The error of the distance queries is $\mathcal{O}(M^2 + n \log n) \cdot (1 + \epsilon)$, where $0 < \epsilon < 1$.

4 Conclusion

In this paper we presented a data structure that answers approximate distance queries in constant time in the case when the input is a set of vertex disjoint Euclidean t -spanners inter-connected with edges of arbitrary weight. The result can be slightly extended and improved. For example, the data structure can be modified to handle the case when each island is a t_i -spanner, i.e., every island has different (although constant) dilation. Also, a refined analysis yields that the data structure of Theorem 1 only uses $\mathcal{O}(|\mathcal{C}|^2 + |\mathcal{V}| \log |\mathcal{V}|)$ space, where \mathcal{C} is the set of all airports.

Recall that the subdivision of the graph into islands needs to be given. An interesting open problem is to find such a subdivision, i.e., compute a subdivision of a graph into a minimum number of vertex-disjoint t -spanners. The problem is NP-hard in the case when $t = 0$, since this is equivalent to finding a minimum clique partition which is known to be NP-hard. For general graphs the problem cannot be approximated within a factor of $|\mathcal{V}|^{1/7+\epsilon}$ in polynomial time [1].

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi. Complexity and Approximation – Combinatorial optimization problems and their approximability properties. Springer Verlag, ISBN 3-540-65431-3, 1999.
2. D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.
3. S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In Proc. *4th European Symposium on Algorithms*, pp. 514-528, 1996.
4. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45(6):891-923, 1998.
5. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.
6. D. Z. Chen. On the all-pairs Euclidean short path problem. In Proc. *6th ACM-SIAM Symposium on Discrete Algorithms*, pp. 292-301, 1995.
7. E. Cohen. Fast algorithms for constructing t -spanners and paths with stretch t . *SIAM Journal on Computing*, 28(1):210–236, 1998.
8. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. In *Numerische Mathematik* vol. 1, 1959.
9. D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.
10. D. Eppstein and D. Hart. Shortest Paths in an Arrangement with k Line Orientations. *10th ACM-SIAM Symposium on Discrete Algorithms*, pp. 310-316, 1999.
11. M. L. Fredman, R. E. Tarjan Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596-615, 1987.
12. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
13. J. Gudmundsson, C. Levkopoulos, G. Narasimhan and M. Smid. Approximate Distance Oracles Revisited. In Proc. *13th International Symposium on Algorithms and Computation*, pp. 357-368, 2002.
14. J. Gudmundsson, C. Levkopoulos, G. Narasimhan and M. Smid. Approximate Distance Oracles for Geometric Spanners. Submitted August 2004. <http://win.tue.nl/hgudmund/glns-adogs-04.pdf>. Extended abstract published in Proc. *13th ACM-SIAM Symposium on Discrete Algorithms*, pp. 828-837, 2002.
15. J. Gudmundsson, G. Narasimhan and M. Smid. Fast Pruning of Geometric Spanners. Submitted June 2004. <http://win.tue.nl/hgudmund/gns-fps-04.pdf>
16. D. Krzrnaric and C. Levkopoulos. Computing hierarchies of clusters from the Euclidean minimum spanning tree in linear time. In Proc. *15th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pp. 443-455, 1995.
17. J. S. B. Mitchell. Shortest paths and networks. In *Handbook of Discrete and Computational Geometry*, pp. 445–466. CRC Press LLC, 1997.
18. R. Raman. Recent results on the single-source shortest paths problem. *SIGACT News* 28:81-87,1997.
19. J. A. Storer and J. H. Reif. Shortest Paths in the Plane with Polygonal Obstacles. *Journal of the ACM*, 41(5): 982-1012, 1994.
20. M. Thorup and U. Zwick. Approximate distance oracles. In Proc. *33rd ACM Symposium on Theory of Computing*, pp. 183-192, 2001.

Multicriteria Global Minimum Cuts

Amitai Armon and Uri Zwick

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel

Abstract. We consider two multicriteria versions of the global minimum cut problem in undirected graphs. In the k -criteria setting, each edge of the input graph has k non-negative costs associated with it. These costs are measured in separate, non interchangeable, units. In the AND-version of the problem, purchasing an edge requires the payment of *all* the k costs associated with it. In the OR-version, an edge can be purchased by paying *any one* of the k -costs associated with it. Given k bounds b_1, b_2, \dots, b_k , the basic multicriteria decision problem is whether there exists a cut C of the graph that can be purchased using a budget of b_i units of the i -th criterion, for $1 \leq i \leq k$.

We show that the AND-version of the multicriteria global minimum cut problem is polynomial for any fixed number k of criteria. The OR-version of the problem, on the other hand, is NP-hard even for $k = 2$, but can be solved in pseudo-polynomial time for any fixed number k of criteria. It also admits an FPTAS. Further extensions, some applications, and multicriteria versions of two other optimization problems are also discussed.

1 Introduction

We consider two multicriteria versions of the global minimum cut problem in undirected graphs. Let $G = (V, E)$ be an undirected graph, and let $w_1, \dots, w_k : E \rightarrow \mathbb{R}^+$ be k nonnegative cost (or weight) functions defined on its edges. A cut C of G is a subset $C \subseteq V$ such that $C \neq \emptyset$ and $C \neq V$. The edges cut by this cut are $E(C) = \{(u, v) \in E \mid u \in C, v \notin C\}$. (As the graph is undirected, C and $V - C$ define the same cut.) In the AND-version of the k -criteria problem, the i -th weight (or cost) of the cut is

$$i\text{-th cost in the AND-version: } w_i(C) = \sum_{e \in E(C)} w_i(e) \quad , \quad 1 \leq i \leq k .$$

In the OR-version of the problem we pay only one of the costs associated with each edge $e \in E(C)$ of the cut. More specifically, we choose a function $\alpha : E(C) \rightarrow \{1, 2, \dots, k\}$ which specifies which cost is paid for each edge of the cut. The i -th cost of the cut C , with respect to the choice function α , is then

$$i\text{-th cost in the OR-version: } w_i(C, \alpha) = \sum_{e \in E(C) \wedge \alpha(e)=i} w_i(e) \quad , \quad 1 \leq i \leq k .$$

The basic multicriteria global minimum cut decision problem asks, given k cost bounds b_1, b_2, \dots, b_k , is there a cut C such that $w_i(C) \leq b_i$, for $1 \leq i \leq k$? In the optimization problem we are given $k-1$ bounds b_1, b_2, \dots, b_{k-1} and asked to find a cut C for which $w_k(C)$ is minimized, subject to the constraints $w_i(C) \leq b_i$, for $1 \leq i \leq k-1$. The version of the problem asks for a cut C for which $\max_{i=1}^k w_i(C)$ is minimized, i.e., a cut whose largest cost is as small as possible. The $P(G, w_1, \dots, w_k) \subseteq \mathbb{R}^k$ of an instance $\langle G, w_1, \dots, w_k \rangle$ is the set of cost vectors of cuts that are not dominated by the cost vector of any other cut. It follows, therefore, that if $(c'_1, c'_2, \dots, c'_k)$ is the cost vector of a cut C' of the graph, then there exists a vector $(c_1, c_2, \dots, c_k) \in P(G, w_1, \dots, w_k)$ such that $c_i \leq c'_i$ for $1 \leq i \leq k$. Corresponding definitions can be made for the OR-version of the problem.

Multicriteria optimization is an active field of research (see, e.g., the books of Climacao [3] and Ehrgott [4]). (Most research is focused, in our terminology, on AND-versions of various optimization problems. All results cited below refer to the AND-versions of the problems, unless stated otherwise.) Papadimitriou and Yannakakis [17] investigated the complexity of several multicriteria optimization problems. In particular, they considered the multicriteria s - t minimum cut problem, in which the cut must separate two specified vertices, s and t . They proved that this problem is strongly NP-complete, even for just two criteria.

We show here that (the AND-version of) the multicriteria global minimum cut decision problem can be solved in polynomial time for any number of criteria, making it strictly easier than its s - t variant. The running time of our algorithm is $O(mn^{2k})$, where $m = |E|$ is the number of edges in the graph, $n = |V|$ is the number of vertices, and k is the number of criteria. This easily implies tractability of the optimization problem and also yields a pseudo-polynomial algorithm for constructing the Pareto set. The problem, however, becomes strongly NP-hard when the number of criteria is not fixed. We also show that the directed version of the problem is strongly NP-hard even for just two criteria.

The single-criterion minimum cut problem has been studied for more than four decades as a fundamental graph optimization problem (see, e.g., [6, 15, 13, 19, 11, 12]). Minimum cuts are used in solving a large variety of problems, including VLSI design, network design and reliability, clustering, and more (see [12] and references therein). The best known deterministic algorithms for this problem run in $O(mn + n^2 \log n)$ time (Nagamochi and Ibaraki [15] and Stoer and Wagner [19]). The best known randomized algorithm runs in $O(m \log^3 n)$ time (Karger [12]). (A huge gap between the deterministic and randomized complexities!) These algorithms are faster than the best known algorithms for the s - t minimum cut problem which are based on network flow.

The polynomial time algorithm for the multicriteria problem relies on the fact that the standard single criterion global minimum cut problem has only a polynomial number of almost optimal solutions. More specifically, Karger and Stein [13] showed that for every $\alpha \geq 1$, not necessarily integral, the number of α -approximate solutions is only $O(n^{2\alpha})$. Karger [12] improved this bound to $O(n^{\lfloor 2\alpha \rfloor})$. Nagamochi [16] gave a deterministic $O(m^2 n + mn^{2\alpha})$ time algo-

rithm for finding all the α -approximate cuts. Our algorithm for the multicriteria problem uses their algorithms.

Apart from the theoretical interest in minimum cuts in the multicriteria setting, there are some applications in which this problem is of interest. (The multicriteria global minimum cut problem is of interest in almost any application of the single criterion global minimum cut problem.) A multicriteria minimum balanced-partition is required, for example, in the situations described in [18].

A special case of the bicriteria global minimum cut problem, called the $\leq r$ problem, was considered by Bruglieri [1, 2]. The input to this problem is an undirected graph $G = (V, E)$ with a single weight function $w : E \rightarrow \mathbb{R}^+$ defined on its edges. The goal is to find a cut of minimum cost that contains at most r edges. This is exactly the optimization version of the bicriteria minimum cut problem, where $w_1(e) = 1$, $w_2(e) = w(e)$, for every $e \in E$, and $w_1(C)$ must not exceed r . Bruglieri [1, 2] ask whether this problem can be solved in polynomial time. We answer their question in the affirmative. We also obtain a polynomial time algorithm for finding a minimum cut which contains at most r vertices on the smallest side of the cut.

As mentioned, most research on multicriteria optimization focused, in our terminology, on AND-versions of various multicriteria optimization problems. We consider here also the OR-versions of the global minimum cuts problem, the shortest path problem, and the minimum spanning tree problem.

OR-versions of multicriteria optimization problems may be seen as generalizations of the scheduling problem on m machines (see [9, 14, 10]). The input to such a scheduling problem is a set of n jobs that should be scheduled on m machines. The i -th job has a cost vector (c_{i1}, \dots, c_{im}) associated with it, where c_{ij} is the processing time of the i -th job on the j -th machine. The goal is to allocate the jobs to the machines so as to minimize the $\max_j C_j$, i.e., the completion time of the last job. (Jobs allocated to the same machine are processed sequentially.) This is precisely the OR-version of the min-max m -criteria minimum cut problem on a graph with two vertices and n parallel edges.

As another example where the OR-version of the multicriteria minimum-cut problem is of interest, consider a cyber-terrorist wishing to disconnect a computer network, where there is more than one option for damaging each link. Specifically, assume that each link can either be disconnected by an electronic attack, which requires a certain amount of work hours (that may differ for different links), or by physically disconnecting it, e.g. by cutting an underground cable using explosives (the required amount may again differ from link to link). Given an upper bound on the amount of available explosives, what is the minimum time required for electronic attacks?

It follows immediately from the simple reduction given above that the OR-version of the multicriteria global minimum cut problem is NP-hard even for just two criteria. We show, however, that the problem can be solved in pseudo-polynomial time for any fixed number of criteria. We also show that the problem can be solved in polynomial time when k , the number of criteria, is fixed and at least $k - 1$ of the weight functions assume only a fixed number of values. We also

obtain some results on the complexity of the OR-versions of the shortest path and minimum spanning tree problems.

The rest of this paper is organized as follows. In the next section we consider the AND-version of the global minimum cut problem. In Section 3 we then consider the OR-version of the problem. In Section 4 we consider the OR-version of the multicriteria shortest path and minimum spanning tree problems. Finally, we conclude in Section 5 with some concluding remarks and open problems.

2 Multicriteria Global Minimum Cut: The AND-Version

We first present a polynomial time algorithm for the min-max version of the multicriteria global minimum cut. The algorithm for solving the min-max version of the problem is then used to solve the decision and optimization problems.

2.1 The Min-Max Problem

An optimal min-max cut is a cut C for which $\max_{i=1}^k w_i(C)$ is minimized. We show that the simple algorithm given in Figure 1 solves the min-max version of the k -criteria global minimum cut problem in polynomial time, for every fixed k . A k -approximate cut in a graph G with respect to a single weight function w' is a cut whose weight is at most k times the weight of the minimum cut.

Theorem 1. *Let G be a graph with n nodes and m edges, and let w_1, \dots, w_k be k weight functions on the edges of G . Then the algorithm in Figure 1 finds an optimal min-max cut in $O(mn^{2k})$ time.*

We begin by proving the correctness of the algorithm. We show that if C is an optimal min-max cut, and D is any other cut in the graph, then $w'(C) \leq k \cdot w'(D)$, where $w'(e) = \sum_{i=1}^k w_i(e)$, for every $e \in E$. This follows as

$$w'(C) = \sum_{i=1}^k w_i(C) \leq k \cdot \max_{i=1}^k w_i(C) \leq k \cdot \max_{i=1}^k w_i(D) \leq k \cdot \sum_{i=1}^k w_i(D) = k \cdot w'(D).$$

The inequality $k \cdot \max_{i=1}^k w_i(C) \leq k \cdot \max_{i=1}^k w_i(D)$ follows from the assumption that C is an optimal min-max cut. In particular, if D is an optimal minimum cut with respect to the single weight function w' , then $w'(C) \leq k \cdot w'(D)$, and it follows that C is a k -approximate cut of G with respect to w' . This proves the correctness of the algorithm.

We next consider the complexity of the algorithm. Karger and Stein [13] showed that every graph has at most $O(n^{2k})$ k -approximate cuts and gave a randomized algorithm for finding an ϵ -approximate representation of them all in $O(n^{2k})$ time. A deterministic algorithm of Nagamochi et al. [16] explicitly finds all the k -approximate cuts in $O(mn^{2k})$ time. Choosing the best min-max cut among all the k -approximate cuts also takes only $O(mn^{2k})$ time. \square

It is also easy to see that for any $1 < \alpha \leq k$, we can find an α -approximate solution to the min-max problem in $O(mn^{2k/\alpha})$ time, by checking all the k/α -approximate cuts in G' .

The randomized algorithm of Karger and Stein [13] extends to finding all the k -approximate minimum r -cuts in $\tilde{O}(n^{2k(r-1)})$ time. (An r -cut is a partition of the graph vertices into r sets, instead of 2). Thus, it is easy to see that the min-max multicriteria problem can also be solved for r -cuts, in $\tilde{O}(mn^{2k(r-1)})$ time using this randomized (Monte-Carlo) algorithm.

2.2 The Decision Problem

We next show that the algorithm for the min-max version of the k -criteria problem can be used to solve the decision version of the problem: Given k bounds b_1, b_2, \dots, b_k , is there a cut C such that $w_i(C) \leq b_i$, for $1 \leq i \leq k$.

Theorem 2. *Given k weight functions $w_1, \dots, w_k : E \rightarrow R^+$ and k bounds b_1, b_2, \dots, b_k , there is an algorithm that runs in $O(mn^{2k})$ time to decide if there is a cut C such that $w_i(C) \leq b_i$, for $1 \leq i \leq k$.*

The decision problem can be easily reduced to the min-max problem. Given k weight functions $w_1, \dots, w_k : E \rightarrow R^+$ and k bounds b_1, b_2, \dots, b_k , we simply produce k new weight functions $w'_i(e) = w_i(e)/b_i$, for every $e \in E$ and $1 \leq i \leq k$, of the weight functions. Clearly the answer to the decision problem is ‘yes’ if and only if there is a cut C for which $\max_{i=1}^k w'_i(C) \leq 1$. □

2.3 The Optimization Problem

We next tackle the optimization problem: Given $k-1$ bounds b_1, b_2, \dots, b_{k-1} , find a cut C for which $w_k(C)$ is minimized, subject to the constraints $w_i(C) \leq b_i$, for $1 \leq i \leq k-1$.

Theorem 3. *Given k weight functions $w_1, \dots, w_k : E \rightarrow R^+$ and $k-1$ bounds b_1, b_2, \dots, b_{k-1} , there is an algorithm that runs in $O(mn^{2k} \log M)$ time to find a cut C for which $w_k(C)$ is minimized, subject to the constraints $w_i(C) \leq b_i$, for $1 \leq i \leq k-1$. Here $M = \sum_{e \in E} w_k(e)$.*

If the k -th weight function assumes only integral values, we can easily use binary search to solve the optimization problem. Given the $k-1$ bounds b_1, b_2, \dots, b_{k-1} , we conduct a binary search for the minimal value b_k for which there is a cut C such that $w_i(C) \leq b_i$, for $1 \leq i \leq k$. As the minimal b_k is an integer in the range $[0, M]$, this requires the solution of only $O(\log M)$ decision problems. □

The algorithm given above is not completely satisfactory as it is not strongly polynomial and does not work with non-integral weights. These problems can be fixed, however, as we show below.

Theorem 4. *Given k weight functions $w_1, \dots, w_k : E \rightarrow R^+$ and $k-1$ bounds b_1, b_2, \dots, b_{k-1} , there is an algorithm that runs in $O(mn^{2k} \log n)$ time to find a cut C for which $w_k(C)$ is minimized, subject to the constraints $w_i(C) \leq b_i$, for $1 \leq i \leq k-1$.*

We first isolate a small interval that contains the minimal value of b_k . Let $S = \{w_k(e) \mid e \in E\}$ be the set of values assumed by the k -th weight function.

The minimum b_k of the optimization problem lies in an interval $[s, ms]$, for some $s \in S$. (If C is the cut that attains the optimum, let s be the weight of the heaviest edge, with respect to w_k , in the cut.) Using a binary search on the values in S , we can find such an interval that contains the minimum. This requires the solution of only $O(\log m) = O(\log n)$ decision problems. Next, we conduct a binary search in the interval $[s, ms]$ until we narrow it down to an interval of the form $[s', (1 + \frac{1}{n})s']$ which is guaranteed to contain the right answer. This again requires the solution of only $O(\log(mn)) = O(\log n)$ decision problems.

Next, we run a modified version of the Min-Max algorithm given in Figure 1 on the following scaled versions of the weights: $w'_i(e) = w_i(e)/b_i$, for $1 \leq i \leq k-1$, and $w'_k(e) = w_k(e)/s'$. It is easy to see that if C is an optimal solution of the optimization problem, then C is also a $(1 + \frac{1}{n})$ -approximate solution of the min-max problem. This in turn implies, as in the proof of Theorem 1, that C is also a $k(1 + \frac{1}{n})$ -approximate minimum cut with respect to the weight function $w'(e) = \sum_{i=1}^k w_i(e)$, for every $e \in E$. Instead of finding all the k -approximate minimum cuts with respect to w' , as done by algorithm Min-Max, we find all the $k(1 + \frac{1}{n})$ -approximate minimum cuts. Among all these cuts we find a cut C for which $w'_i(C) \leq 1$, i.e., $w_i(C) \leq b_i$, for $1 \leq i \leq k-1$, and for which $w_k(C)$ is minimized. This cut is the optimal solution to the optimization problem.

We next analyze the complexity of the algorithm. The $O(\log n)$ decision problems can be solved in $O(mn^{2k} \log n)$ time. All the $k(1 + \frac{1}{n})$ -approximate cuts can then be found in $O(mn^{2k(1 + \frac{1}{n})}) = O(mn^{2k})$ time using the algorithm of Nagamochi [16]. (Note that $n^{1/n} = O(1)$.) Checking all these cuts also takes only $O(mn^{2k})$ time. This completes the proof of the theorem. \square

Algorithm Min-Max($G(V, E, w_1, \dots, w_k)$):

1. Let $w'(e) = \sum_{i=1}^k w_i(e)$, for every $e \in E$.
2. Find all the k -approximate minimum cuts in G with respect to w' .
3. Among all the cuts C found in the previous step find the one for which $\max_{i=1}^k w_i(C)$ is minimized.

Fig. 1. A strongly polynomial time algorithm for the min-max version of the k -criteria global minimum cut problem

2.4 Two Applications

Theorem 5. Let $G = (V, E)$ be a graph with n vertices and m edges, and let $w : E \rightarrow \mathbb{R}^+$ be a weight function. For any integer r with $1 \leq r \leq m$, there is an algorithm that runs in time $O(mn^4 \log n)$ and finds a cut C such that $w(C) \leq r$.

We simply let $w_1(e) = 1$ and $w_2(e) = w(e)$, for every $e \in E$, and solve the optimization problem with $b_1 = r$. \square

As mentioned in the introduction, this solves an open problem raised by Bruglieri [1, 2]. We also have:

Theorem 6. *Let $G = (V, E)$ be a complete graph with n vertices and $w : E \rightarrow \mathbb{R}^+$ a weight function. For every $1 \leq r \leq n$, there exists a cut C such that $w_1(C) \leq r$ and $w_2(C) \leq O(n^6 \log n)$.*

We set up two weight functions over a complete graph on $n = |V|$ vertices: $w_1(u, v) = 1$, for every $u, v \in V$, and $w_2(u, v) = w(u, v)$, if $(u, v) \in E$, and $w(u, v) = 0$, otherwise. We then find a cut C that minimizes $w_2(C)$ subject to the constraint $w_1(C) \leq r(n - r)$. \square

2.5 The Pareto Set

Suppose all weight functions are integral. Let $M_i = \sum_{e \in E} w_i(e)$, for $1 \leq i \leq k$. The Pareto set can be trivially found by invoking the basic decision algorithm $\Pi_{i=1}^k M_i$ times, or the optimization algorithm $\Pi_{i=1}^{k-1} M_i$ times. These naive algorithms are pseudo-polynomial for every fixed k .

Using very similar ideas we can also obtain an FPTAS for finding an ϵ -approximate Pareto set, a notion defined by Papadimitriou and Yannakakis [17]. It is defined as a set of feasible k -tuples, such that for every solution there is a k -tuple in the set within a factor of $(1 - \epsilon)$ in all coordinates. More formally, the set $P_\epsilon(G, w_1, \dots, w_k)$ is a set of cost vectors of cuts in the graph such that for every cut C there exists $(c_1, \dots, c_k) \in P_\epsilon(G, w_1, \dots, w_k)$ such that $(1 - \epsilon)c_i \leq w_i(C)$ for $1 \leq i \leq k$. It is easy to see that we can find this set in polynomial time by checking only powers of $1 - \epsilon$, instead of checking all the possible values.

2.6 Hardness Results

Theorem 7. *Let $G = (V, E)$ be a complete graph with n vertices and $w : E \rightarrow \mathbb{R}^+$ a weight function. For every $1 \leq r \leq n$, there exists a cut C such that $w_1(C) \leq r$ and $w_2(C) \leq O(n^6 \log n)$.*

We use a reduction from the ϵ -approximate Pareto set problem (see [5], problem ND17): Given an unweighted input graph $G = (V, E)$ on $n = 2r$ vertices and a bound b , is there a bisection of the graph that cuts at most b edges? We transform such an instance in the following way: Assume that $V = \{1, 2, \dots, n\}$. We add two vertices, s and t , and add edges connecting them to each of the vertices in V . Let $G' = (V', E')$ be the resulting graph. Each edge of G' is now assigned $n + 3$ weights. For $1 \leq i \leq n$, we let $w_i(s, i) = w_i(t, i) = 1$, and $w_i(e) = 0$ for all other edges. We assign $w_{n+1}(e) = 1$ for the edges of the form (s, i) , $i \in V$, and $w_{n+1}(e) = 0$, otherwise. Similarly, we assign $w_{n+2}(e) = 1$ for the edges of the form (t, i) , $i \in V$, and $w_{n+2}(e) = 0$, otherwise. Finally, $w_{n+3}(e) = 1$ for $e \in E$, and $w_{n+3}(e) = 0$, otherwise. It is now easy to see that G has a bisection of width

at most b if and only if G' has a cut C for which $w_i(C) \leq 1$, for $1 \leq i \leq n$, $w_{n+1}(C), w_{n+2}(C) \leq r$, and $w_{n+3}(C) \leq b$. \square

It is also not difficult to show that the multicriteria global minimum cut problem is strongly NP-complete even for two criteria. We omit the details.

3 Multicriteria Global Minimum Cut: The OR-Version

3.1 Relation to Scheduling on Unrelated Machines

As mentioned in the introduction, there is a trivial reduction from the scheduling on unrelated machines problem to the OR-version of the min-max multicriteria global minimum cut problem. Known hardness results for the scheduling problem (see Lenstra [14]) then imply the following:

Theorem 8. *Let $G = (V, E)$ be a graph with n nodes and m edges, and let $w_1, \dots, w_k : E \rightarrow \mathbb{N}$ be k weight functions. Then there is a polynomial time algorithm that finds a cut C of G such that $\max_{i=1}^k w_i(C) \leq 3/2 \cdot \min_{C'} \max_{i=1}^k w_i(C')$.*

The scheduling problem on a fixed number of unrelated machines can however be solved in pseudo-polynomial time. Horowitz and Sahni [9] present a simple branch-and-bound pseudo-polynomial algorithm for that problem which runs in $O(m^2(kM)^{k-1})$ time, where m is the number of jobs, k is the number of machines, and M is the optimal makespan. This immediately implies:

Theorem 9. *Let $G = (V, E)$ be a graph with n nodes and m edges, and let $w_1, \dots, w_k : E \rightarrow \mathbb{N}$ be k weight functions. Then there is a polynomial time algorithm that finds a cut C of G and a choice function $\alpha : E(C) \rightarrow \{1, 2, \dots, k\}$ such that $\max_{i=1}^k w_i(C, \alpha) \leq m \cdot \min_{C'} \max_{i=1}^k w_i(C')$.*

Jansen and Porkolab [10] obtained an FPTAS for the unrelated machines scheduling problem, which runs in $O(m(k/\epsilon)^{O(k)})$ time. It can be used instead of the exact algorithm of [9] when approximate solutions are acceptable.

3.2 The Min-max Version

We show that the simple algorithm given in Figure 2, which is a variant of the algorithm given in Figure 1, solves the OR-version of the min-max problem in pseudo-polynomial time, for any fixed number of criteria.

Theorem 10. *Let $G = (V, E)$ be a graph with n nodes and m edges, and let $w_1, \dots, w_k : E \rightarrow \mathbb{N}$ be k weight functions. Then there is a polynomial time algorithm that finds a cut C of G such that $\max_{i=1}^k w_i(C) \leq O(m^2 n^{2k} (kM)^{k-1})$, where $M = \max_{i=1}^k \max_{e \in E} w_i(e)$.*

We begin again with the correctness proof. Let C be an optimal min-max cut and let α be the corresponding optimal choice function. Let D be any other

Algorithm Min-Max-Or($G(V, E, w_1, \dots, w_k)$):

1. Let $w'(e) = \min_{i=1}^k w_i(e)$, for every $e \in E$.
2. Find all the k -approximate minimum cuts in G with respect to w' .
3. For each of the cuts C found in the previous step, find the best choice function $\alpha : E(C) \rightarrow \{1, 2, \dots, k\}$.
4. Output the best cut and choice function found.

Fig. 2. A pseudo-polynomial time algorithm for the min-max version of the k -criteria global minimum cut problem

cut. We show that $w'(C) \leq k \cdot w'(D)$, where $w'(e) = \min_{i=1}^k w_i(e)$, for every $e \in E$. To see that, we let $\beta : E(D) \rightarrow \{1, 2, \dots, k\}$ be a choice function for which $\beta(e) = i$ if $w_i(e) \leq w_j(e)$, for every $1 \leq j \leq k$. Then,

$$w'(C) \leq k \cdot \max_{i=1}^k w_i(C, \alpha) \leq k \cdot \max_{i=1}^k w_i(D, \beta) \leq k \cdot w'(D).$$

The second inequality follows as (C, α) is an optimal solution of the min-max problem.

We next consider the complexity of the algorithm. The k -approximate cuts with respect to w' can be found again in $O(mn^{2k})$ time using the algorithm of Nagamochi [16]. For each one of the $O(n^{2k})$ approximate cuts produced, we find an optimal choice function using the algorithm of Horowitz and Sahni [9]. The total running time is then $O(mn^{2k} + n^{2k} \cdot m^2(kM)^{k-1}) = O(mn^{2k} + m^2 n^{2k} (kM)^{k-1})$, where M is the value of the optimal solution. \square

Theorem 11. *Let $G(V, E, w_1, \dots, w_k)$ be a graph with n nodes and m edges, and let M be the value of the optimal solution. Then, the min-max version of the k -criteria global minimum cut problem can be solved in $O(mn^{2k} + m^2 n^{2k} (kM)^{k-1})$ time.*

The proof is identical to the proof of Theorem 10 with the exact algorithm of Horowitz and Sahni [9] replaced by the FPTAS of Jansen and Porkolab [10]. \square

As in Section 2, we can use the algorithm for the min-max version of the problem to solve the decision and optimization versions of the problem. We omit the obvious details.

3.3 A Case That Can be Solved in Polynomial Time

We now discuss a restriction of the min-max problem that can be solved in strongly polynomial time. For simplicity, we consider the bicriteria problem.

Theorem 12. *Let $G(V, E, w_1, w_2)$ be a graph with n nodes and m edges, and let r be a positive integer. Then, the min-max version of the bicriteria global minimum cut problem can be solved in $O(m^{r+1}n^4)$ time.*

Assume, without loss of generality, that w_2 assumes only r different real values a_1, a_2, \dots, a_r . Let $E_i = w_2^{-1}(a_i) = \{e \in E \mid w_2(e) = a_i\}$, for $1 \leq i \leq r$. Consider an optimal min-max cut C and an optimal choice function $\alpha : E(C) \rightarrow \{1, 2\}$ for it. It is easy to see that for every $1 \leq i \leq r$ there is a threshold t_i such that if $e \in E_i$, then $\alpha(e) = 1$ if and only if $w_1(e) \leq t_i$. (Indeed, if there are two edges $e_1, e_2 \in E_i$ such that $w_1(e_1) < w_1(e_2)$, $\alpha(e_1) = 2$ and $\alpha(e_2) = 1$, then the choice function α' which reverses the choices of α on e_1 and e_2 is a better choice function. We assume here, for simplicity, that all weights are distinct.) As there are at most $m + 1$ essentially different thresholds for each set E_i , the total number of choice functions that should be considered is only $O(m^r)$. With a given choice function $\alpha : E \rightarrow \{1, 2\}$, the problem reduces to an AND-version of the problem with the weights $w'_i(e) = w_i(e)$, if $\alpha(e) = i$, and $w'_i(e) = 0$, otherwise, for $i = 1, 2$. As each such problem can be solved in $O(mn^4)$ time, the total running time of the resulting algorithm is $O(m^{r+1}n^4)$. \square

4 OR-Version of Other Multicriteria Problems

In this section we consider the OR-versions of the bicriteria shortest path and minimum spanning tree problems. Our results can probably be extended to any fixed number of criteria.

4.1 Shortest Paths

The input to the problem is a directed graph $G = (V, E)$ with two weight functions $w_1, w_2 : E \rightarrow \mathbb{R}^+$ defined on its edges, two vertices $s, t \in V$, and two bounds b_1, b_2 . The question is whether there is a path P from s to t in the graph and a choice function $\alpha : P \rightarrow \{1, 2\}$ such that $w_1(\alpha^{-1}(1)) \leq b_1$ and $w_2(\alpha^{-1}(2)) \leq b_2$. (Recall the subway example given in the introduction.)

It is easy to see, using a simple reduction from the scheduling on unrelated machines problem, that the OR-version of the bicriteria shortest path problem is NP-hard. We show, however, that it can be solved in pseudo-polynomial time. A FPTAS for the problem is easily obtained by scaling.

Theorem 13.

Let $G = (V, E)$ be a directed graph with n vertices and m edges. Let $w_1, w_2 : E \rightarrow \mathbb{R}^+$ be two weight functions. Let $b_1, b_2 \in \mathbb{R}^+$ be two bounds. Let $W = \max_{e \in E} w_1(e)$.

The OR-version of the problem can be easily reduced to the AND-version of the problem by replacing each edge e having a weight vector $(w_1(e), w_2(e))$ by two parallel edges e' and e'' having weight vectors $(w_1(e), 0)$ and $(0, w_2(e))$. The standard, AND-version, of the problem can be solved using an algorithm of Hansen [7] within the claimed time bound. \square

4.2 Minimum Spanning Trees

Next we consider the OR-version of the bicriteria minimum spanning tree problem. The input is an undirected graph $G = (V, E)$, two weight functions $w_1, w_2 :$

$E \rightarrow \mathbb{R}$, and two bounds b_1 and b_2 . The question is whether there exist a spanning tree T and a choice function $\alpha : T \rightarrow \{1, 2\}$ such that $w_1(\alpha^{-1}(1)) \leq b_1$ and $w_2(\alpha^{-1}(2)) \leq b_2$.

The OR-version of the bicriteria minimum spanning tree problem is again easily seen to be NP-hard. We provide a polynomial time algorithm for a special case of the problem, and a pseudo-polynomial time algorithm for the general case.

Theorem 14.

Let $G = (V, E)$ be a graph with n vertices and m edges, and let $c, b_1, b_2 \in \mathbb{R}$ with $b_1 < b_2$. Let $w_1, w_2 : E \rightarrow \mathbb{R}$ be two weight functions such that $w_1(e) \leq c$ and $w_2(e) = c$ for all $e \in E$.

We simply solve the standard minimum spanning tree problem with respect to the weight function w_1 and obtain a minimum spanning tree T . For the $\lfloor b_2/c \rfloor$ heaviest edges of T we choose to pay the w_2 cost, and for all the others we pay the w_1 cost. The correctness of this procedure follows from the well known fact that if the weights of the edges of T are $a_1 \leq a_2 \leq \dots \leq a_{n-1}$, and if T' is any other spanning tree of the graph G with edge weights $a'_1 \leq a'_2 \leq \dots \leq a'_{n-1}$, then $a_i \leq a'_i$, for $1 \leq i \leq n - 1$. □

Theorem 15.

Let $G = (V, E)$ be a graph with n vertices and m edges, and let $b_1, b_2 \in \mathbb{R}$ with $b_1 < b_2$. Let $w_1, w_2 : E \rightarrow \mathbb{R}$ be two weight functions such that $w_1(e) \leq b_1$ and $w_2(e) \leq b_2$ for all $e \in E$. Then the OR-version of the problem can be solved in time $O(n^4 b_1 b_2 \log(b_1 b_2))$.

The OR-version of the problem can be easily reduced to the AND-version of the problem by replacing each edge e having a weight vector $(w_1(e), w_2(e))$ by two parallel edges e' and e'' having weight vectors $(w_1(e), 0)$ and $(0, w_2(e))$. The standard, AND-version, of the problem can be solved using an algorithm of Hong [8] within the claimed time bound. □

5 Concluding Remarks

We showed that the standard (i.e., the AND-version) multicriteria global minimum cut problem can be solved in polynomial time for any fixed number k of criteria. The running time of our algorithm, which is $O(mn^{2k})$, is fairly high, even for a small number of criteria. Improving this running time is an interesting open problem. We also considered the OR-version of the problem and showed that it is NP-hard even for just two criteria. It can be solved, however, in pseudo-polynomial time, and it also admits an FPTAS, for any fixed number of criteria. Finally, we considered the OR-versions of the bicriteria shortest path and minimum spanning tree problems, and showed that both of them are NP-hard but can be solved in pseudo-polynomial time. It will also be interesting to study OR-versions of other multicriteria optimization problems.

References

1. M. Bruglieri, M. Ehrgott, and H.W. Hamacher. Some complexity results for k -cardinality minimum cut problems. Technical Report 69/2000, Wirtschaftsmathematik, University of Kaiserslautern, 2000.
2. M. Bruglieri, F. Maffioli, and M. Ehrgott. Cardinality constrained minimum cut problems: Complexity and algorithms. *Discrete Applied Mathematics*, 137(3):311–341, 2004.
3. J. Climacao. *Multicriteria Analysis*. Springer-Verlag, 1997.
4. M. Ehrgott. *Multicriteria optimization*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 2000.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
6. R.E. Gomory and T. C. Hu. Multiterminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.
7. P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making: Theory and Applications, LNEMS 177*, pages 109–127. Springer-Verlag, Berlin, 1980.
8. S.P. Hong, S.J. Chung, and B.H. Park. A fully-polynomial bicriteria approximation scheme for the constrained minimum spanning tree problem. *Operations Research Letters*, 32(3):233–239, 2004.
9. E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non-identical processors. *Journal of the ACM*, 23:317–327, 1976.
10. K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *Proceedings of the thirty-first annual ACM Symposium on Theory of Computing*, pages 408–417, 1999.
11. D. R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.
12. D. R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, 2000.
13. D. R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, 1996.
14. J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
15. H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.
16. H. Nagamochi, K. Nishimura, and T. Ibaraki. Computing all small cuts in an undirected network. *SIAM J. Discrete Mathematics*, 10(3):469–481, 1997.
17. C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *IEEE Symposium on Foundations of Computer Science*, pages 86–92, 2000.
18. K. Schloegel, G. Karypis, and V. Kumar. A new algorithm for multi-objective graph partitioning. In *European Conference on Parallel Processing*, pages 322–331, 1999.
19. M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

Polyline Fitting of Planar Points Under Min-Sum Criteria

Boris Aronov^{1,*}, Tetsuo Asano^{2,**}, Naoki Katoh³,
Kurt Mehlhorn⁴, and Takeshi Tokuyama⁵

¹ Polytechnic University
aronov@cis.poly.edu

² Japan Advanced Institute of Science and Technology
t-asano@jaist.ac.jp

³ Kyoto University
naoki@archi.kyoto-u.ac.jp

⁴ Max-Planck-Institut für Informatik
mehlhorn@mpi-sb.mpg.de

⁵ Tohoku University
tokuyama@dais.is.tohoku.ac.jp

Abstract. Fitting a curve of a certain type to a given set of points in the plane is a basic problem in statistics and has numerous applications. We consider fitting a polyline with k joints under the min-sum criteria with respect to L_1 - and L_2 -metrics, which are more appropriate measures than uniform and Hausdorff metrics in statistical context. We present efficient algorithms for the 1-joint versions of the problem, and fully polynomial-time approximation schemes for the general k -joint versions.

1 Introduction

Curve fitting aims to approximate a given set of points in the plane by a curve of a certain type. This is a fundamental problem in statistics, and has numerous applications. In particular, it is a basic operation in *regression analysis*. *Linear regression* approximates a point set by a line, while *non-linear regression* approximates it by a non-linear function from a given family.

In this paper, we consider the case where the points are fitted by a polygonal curve (*polyline*) with k joints, see Figure 1. This is often referred to as *polygonal approximation* or *polygonal fitting* problem. It is used widely. For example, it is commonly employed in scientific and business analysis to represent a data set by a polyline with a small number of joints. The best representation is the polyline minimizing the error of approximation. Error is either defined as the maximum (vertical) distance of any

* Work of B.A. on this paper was supported in part by NSF ITR Grant CCR-00-81964. Part of the work was carried out while B.A. was visiting JAIST.

** Work of T.A. was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B).

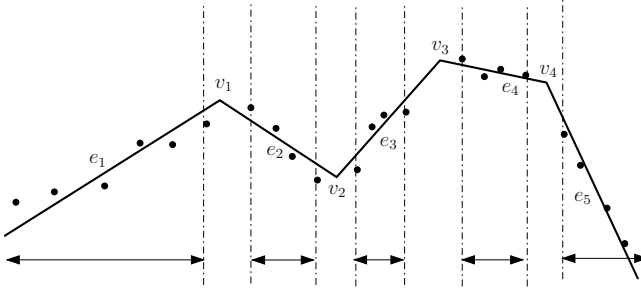


Fig. 1. A 4-joint polyline fitting a set of points

input point from the polyline (*min-max-optimization*) or the sum of vertical distances (*min-sum-approximation*). In either case, distance is measured in some norm. We follow common practice and restrict ourselves to norms L_1 and L_2 .

In this paper, we focus on the L_1 - and L_2 -fitting problems when the desired curve is a k -joint *polyline*; in other words, it is a continuous piecewise-linear x -monotone curve with $k + 1$ linear components. We assume that a coordinate system is fixed, and the input points are sorted with respect to their x -coordinate values. To the authors' knowledge, the computational complexity of the optimal k -joint problem under either of these minimization criteria has not been previously investigated. More specifically, it seems that an efficient solution of the L_1 -fitting problem extending the result of Imai *et al.* [8] is theoretically challenging even for the 1-joint problem.

In this paper, we begin by considering the 1-joint problem. We give algorithms of complexity $O(n)$ and $\tilde{O}(n^{4/3})$ time for the L_2 and L_1 criteria, respectively.¹ The L_2 -fitting algorithm is simple and practical, whereas the L_1 -fitting algorithm depends on using a semi-dynamic range search data structure and parametric search. For general k , we present two approximation schemes. Let z_{opt} be the minimum fitting error for a k -joint polyline and let ϵ be a positive constant. We give a polynomial-time approximation scheme (PTAS) to compute a $\lfloor (1 + \epsilon)k \rfloor$ -joint fitting whose error is at most z_{opt} and we describe a fully polynomial-time approximation scheme (FPTAS) to compute a k -joint polyline with $(1 + \epsilon)z_{\text{opt}}$ fitting error, and consequently show that the problems cannot be strongly NP-hard, although their NP-hardness remains open.

2 Preliminaries

A k -joint polyline is an alternating sequence $P = (e_1, \mathbf{v}_1, e_2, \mathbf{v}_2, \dots, e_k, \mathbf{v}_k, e_{k+1})$ of line segments (*links*) and joint vertices (*joints*), where e_s and e_{s+1} share the endpoint \mathbf{v}_s , for $s = 1, 2, \dots, k$, and e_1 and e_{k+1} are infinite rays. We denote the link e_s on line $y = a_s x - b_s$ by (a_s, b_s) if the interval of the values of x corresponding to the link is understood. A joint \mathbf{v}_s is represented by the pair (u_s, v_s) of its coordinate values. Thus,

¹ We write $f(n) = \tilde{O}(g(n))$ if there exists an absolute constant $c \geq 0$ such that $f(n) = O(g(n) \log^c n)$.

the connectivity and monotonicity of the polyline can be guaranteed by requiring that $v_s = a_s u_s - b_s = a_{s+1} u_s - b_{s+1}$, for $s = 1, 2, \dots, k+1$, and $u_1 < \dots < u_k$.

We now formulate the problem of fitting a k -joint polyline to an n -point set. Given a set of points $S = \{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \dots, p_n = (x_n, y_n)\}$ with $x_1 < x_2 < \dots < x_n$ and an integer k , and setting $u_0 = -\infty$ and $u_{k+1} = \infty$ for convenience, find a polyline $P = ((a_1, b_1), (u_1, v_1), (a_2, b_2), (u_2, v_2), \dots, (u_k, v_k), (a_{k+1}, b_{k+1}))$ minimizing one of the following three quantities for L_1 -, L_2 -, and uniform metric fitting, respectively:

$$L_1: \sum_{s=1}^{k+1} \sum_{u_{s-1} < x_i \leq u_s} |a_s x_i - b_s - y_i|, \quad (1)$$

$$L_2: \sum_{s=1}^{k+1} \sum_{u_{s-1} < x_i \leq u_s} (a_s x_i - b_s - y_i)^2, \quad (2)$$

$$\text{Uniform metric: } \max_{s=1, \dots, k+1} \left\{ \max_{u_{s-1} \leq x_i \leq u_s} |a_s x_i - b_s - y_i| \right\}. \quad (3)$$

For $k = 0$, the problems are linear regression problems. The L_2 -linear regression is well known as the *Gaussian least-squares method*. Once we compute $A_n = \sum_{i=1}^n x_i$, $B_n = \sum_{i=1}^n y_i$, $C_n = \sum_{i=1}^n x_i^2$, $D_n = \sum_{i=1}^n x_i^2$, and $E_n = \sum_{i=1}^n x_i y_i$ in linear time, we can construct an optimal fitting line $y = ax - b$ by considering the partial derivatives of the objective function and solving a 2×2 system of linear equations. The linear regression problem with respect to the uniform error is to find a pair of parallel lines at the minimum vertical distance that contain all the given points between them. This can be done by applying the *rotating caliper method* that computes antipodal pairs of points on the convex hull of the point set. For an x -sorted point set this can be done in $O(n)$ time [15]. The L_1 -linear regression problem is more involved; however, a linear-time algorithm has been devised by Imai *et al.* [8] based on Megiddo's prune-and-search paradigm.

3 Fitting a 1-Joint Polyline

We consider the problem of fitting a 1-joint polyline to a set of points. We proceed in two steps. We first assume that the joint vertex lies in a fixed interval $[x_q, x_{q+1}]$ and later eliminate this assumption. Let $S_1(q) = \{p_1, p_2, \dots, p_q\}$ and $S_2(q) = \{p_{q+1}, \dots, p_n\}$. Our objective polyline consists of two links lying on lines $\ell_1: y = a_1 x - b_1$ and $\ell_2: y = a_2 x - b_2$, respectively. We call a tuple (a_1, b_1, a_2, b_2) *feasible* if the two lines $y = a_1 x - b_1$ and $y = a_2 x - b_2$ meet at a point whose x -coordinate $u = \frac{b_1 - b_2}{a_1 - a_2}$ lies in the interval $[x_q, x_{q+1}]$. Our goal here is to find a feasible tuple (a_1, b_1, a_2, b_2) representing a 1-joint polyline minimizing

$$\sum_{i=1}^q |a_1 x_i - b_1 - y_i| + \sum_{i=q+1}^n |a_2 x_i - b_2 - y_i| \quad \text{and} \quad (4)$$

$$\sum_{i=1}^q (a_1 x_i - b_1 - y_i)^2 + \sum_{i=q+1}^n (a_2 x_i - b_2 - y_i)^2, \quad (5)$$

for L_1 - and L_2 -fitting, respectively. Minimizing (4) is equivalent to, provided $a_1 \neq a_2$, minimizing $\sum_{i=1}^n w_i$ subject to

$$\begin{aligned} -w_i &\leq a_1 x_i - b_1 - y_i \leq w_i, & \text{for } i \leq q, \\ -w_i &\leq a_2 x_i - b_2 - y_i \leq w_i, & \text{for } i \geq q+1, \quad \text{and} \\ x_q &\leq \frac{b_1 - b_2}{a_1 - a_2} \leq x_{q+1}, \end{aligned} \quad (6)$$

where the last line represents the feasibility condition.

Lemma 1. *For either L_1 - or L_2 -fitting criterion, the 1-joint problem for a fixed q reduces to solving two convex programming problems.*

We omit the proof due to space limitations. From the above lemma, it is clear that the optimal 1-joint polyline can be computed by using linear/quadratic programming. However, we aim to design combinatorial algorithms for these problems. Indeed, we can classify the solution into two types: (1) The joint is either on the line $x = x_q$ or $x = x_{q+1}$. (2) The joint lies strictly in the interior of the interval $[x_q, x_{q+1}]$. We call the solution *fixed* in the former case and *free* otherwise. We now have the following simple observation.

Lemma 2. *If the solution is fixed, the joint is located on either of the two vertical lines $x = x_q$, $x = x_{q+1}$.*

If the joint is on the line $x = x_{q+1}$, we can regard it as a solution for the partition into $S_1(q+1) = S_1(q) \cup \{p_{q+1}\}$ and $S_2(q+1) = S_2(q) \setminus \{p_{q+1}\}$. Thus, for each partition, we essentially need to solve two subproblems: (1) the free problem and (2) the fixed problem where the joint is on the vertical line $x = x_q$. This leads to the following generic algorithm: For each partition of S into two intervals S_1 and S_2 , we first consider the free problem ignoring the feasibility constraint, and check whether the resulting solution is feasible or not, i.e., we verify that the intersection point lies in the strip between p_q and p_{q+1} . If it is feasible, it is the best solution for the partition. Otherwise, we consider the fixed solution adding the constraint that the joint lie on $x = x_q$, and report the solution for the partition. After processing all $n - 1$ possible partitions, we report the solution with the smallest error.

If it takes $O(f(n))$ time to process a subproblem for each partition, the total time complexity is $O(nf(n))$. For efficiency, we design a dynamic algorithm to process each partition so that $f(n)$ is reduced in the amortized sense.

3.1 The L_2 1-Joint Problem

We show how to construct an optimal L_2 -fitting 1-joint polyline in linear time. We process the partitions $(S_1(q), S_2(q))$ starting from $q = 1$ to $q = n - 1$, in order. We maintain the sums, variances, and covariances $A_q = \sum_{i=1}^q x_i$, $B_q = \sum_{i=1}^q y_i$, $C_q = \sum_{i=1}^q x_i^2$, $D_q = \sum_{i=1}^q y_i^2$, and $E_q = \sum_{i=1}^q x_i y_i$ incrementally, at constant amortized cost. They also provide us with the corresponding values for $S_2(q)$ if we precompute those values for S , i.e., $\sum_{i=q+1}^n x_i = A_n - A_q$ etc.

For the free case, the objective function is separable, in the sense that the optimal solution can be identified by finding (a_1, b_1) minimizing $\sum_{i=1}^q (a_1 x_i - b_1 - y_i)^2$ and (a_2, b_2) minimizing $\sum_{j=q+1}^n (a_2 x_j - b_2 - y_j)^2$ independently. Each can be computed in $O(1)$ time from the values of A_q, \dots, E_q as explained in section 2. The feasibility check of the solution is done in $O(1)$ time by computing the intersection point of the corresponding pair of lines. It remains to solve the subproblems with the additional constraint that the joint is at $x = x_q$. Put

$$f(a_1, b_1, a_2, b_2) = \sum_{i=1}^q (a_1 x_i - b_1 - y_i)^2 + \sum_{j=q+1}^n (a_2 x_j - b_2 - y_j)^2, \quad (7)$$

$$g(a_1, b_1, a_2, b_2) = a_1 x_q - b_1 - a_2 x_q + b_2, \text{ and} \quad (8)$$

$$L(a_1, b_1, a_2, b_2) = f(a_1, b_1, a_2, b_2) - \lambda g(a_1, b_1, a_2, b_2), \quad (9)$$

so that $f(\cdot)$ is the function to be minimized and the joint constraint can be expressed as $g(\cdot) = 0$. Now, a standard Lagrange multiplier method solves the problem, and we have a linear equation whose coefficients can be expressed in terms of x_q, A_q, \dots, E_q . Thus, we have the following

Theorem 1. *L_2 -optimal 1-joint fitting can be computed in linear time.*

3.2 The L_1 1-Joint Problem

Semi-dynamic L_1 Linear Regression. We start with the problem of computing the optimal linear L_1 -fitting (i.e., linear regression) of the input point set, i.e., we seek the line $\ell_{\text{opt}}: y = ax - b$ minimizing $\sum_{i=1}^n |ax_i - b - y_i|$.

The difficulty with the L_1 -fitting problem is that, written in linear programming terms (as in (6)), it has $n + 2$ variables, in contrast to the least-squares case where the problem is directly solved as a bivariate problem. Nonetheless, Imai *et al.* [8] devised an optimal linear-time algorithm for computing ℓ_{opt} based on the multidimensional prune-and-search paradigm using the fact that the optimal line bisects the point set. In order to design an efficient algorithm for the 1-joint fitting problem, we consider a semi-dynamic version of the L_1 linear regression for a point set P with low amortized time complexity, where we dynamically maintain P with insertions and deletions under an assumption that P is always a subset of a fixed universe S of size n that is given from the outset. (In fact, for our application, it is sufficient to be able to start with $P = \emptyset$ and handle only insertions, and to start with $P = S$ and handle only deletions. Moreover, the order of insertions and deletions is known in advance. The data structure we describe below is more general.)

Consider the dual space, with $p_i = (x_i, y_i)$ transformed to the dual line $Y = f_i(X)$ where $f_i(X) = x_i X - y_i$. The line $y = ax - b$ is transformed to the point (a, b) in the dual space. The k th level of the arrangement $\mathcal{A} = \mathcal{A}(S^*)$ of the set S^* of dual lines is the trajectory of the k th largest value among $f_i(X)$.² We call the $\lceil n/2 \rceil$ th level the *median level*.

² We use an asterisk to denote geometric dual of a point, line, or a set of lines/points.

Lemma 3 (Imai et al. [8]). *If the optimal L_1 -fitting line is given by $y = a_{\text{opt}}x - b_{\text{opt}}$, its dual point $(a_{\text{opt}}, b_{\text{opt}})$ is on the median level if n is odd, and between the $\frac{n}{2}$ th level and the $(\frac{n}{2} + 1)$ th level if n is even.*

Now, given X -value t , consider the point $(t, f_i(t))$ for each $i = 1, 2, \dots, n$, and let $F(t)$ be the sum of the $\lfloor n/2 \rfloor$ largest values in $\{f_i(t) : i = 1, 2, \dots, n\}$ and $G(t)$ be the sum of the $\lfloor n/2 \rfloor$ smallest values in the same set. Put $H(t) = F(t) - G(t)$. $H(t)$ gives the L_1 fitting error of the dual line of any point (t, y) on the median level (or between the two median levels if n is even). Thus, by Lemma 3, $H(t)$ is minimized at $t = a_{\text{opt}}$.

Lemma 4. *$F(t)$ is a convex function, while $G(t)$ is concave. As a consequence, $H(t)$ is also convex. $H(t)$ has either slope 0 at $t = a_{\text{opt}}$ or its slope changes from negative to positive at $t = a_{\text{opt}}$.*

Proof. The convexity follows directly from the fact that, in any line arrangement, the portions of the lines lying on or below (resp. on or above) any fixed level k can be decomposed into k non-overlapping concave (resp. convex) chains; see, for example, [2].

Suppose a fixed universe S^* of lines is given. We need a data structure that maintains a subset $P^* \subseteq S^*$ and supports the following operations:

Median-Location Query. For a query value t , return the point on the $\lfloor n/2 \rfloor$ th highest line at $X = t$.

Slope-Sum Query. For a query point $p = (t, y)$, return the sum of the slopes of lines below p at $X = t$.

Height-Sum Query. For a query value $p = (t, y)$, return the sum of the Y -coordinates of the lines below p at $X = t$. The height-sum query reduces to a slope-sum query plus a constant-term-sum query.

Update. A line in S^* is added to or removed from P^* .

Suppose a data structure supporting such queries on a set $P^* \subseteq S^*$ of lines in $O(\tau(n))$ time is available, where $n = |S^*|$. Then we can query the slopes of F and G at t , and hence compute the slope of H at t in $O(\tau(n))$ time. Because of convexity of H , we have the following:

Lemma 5. *Given t , we can decide whether $t < a_{\text{opt}}$, $t > a_{\text{opt}}$, or $t = a_{\text{opt}}$ in $O(\tau(n))$ time.*

Thus, we can perform binary search to find a_{opt} . We show below how to make this search strongly polynomial. Once we know a_{opt} , we determine b_{opt} by the median-location query at $t = a_{\text{opt}}$.

Semi-dynamic Data Structure for the Queries. We show how to realize semi-dynamic median-location query and sum-queries. As a preliminary step, we describe a semi-dynamic data structure for vertical ray queries, i.e., queries of the form: Given a vertical upward ray starting at (t, z) determine the number of lines in P^* intersected by the ray, the sum of their slopes, and the sum of their constant terms. A dual line $Y = x_iX - y_i$ is above (t, z) iff the primal point (x_i, y_i) is above the line $y = tx - z$. Thus our problems

reduce to half-space queries in the primal plane. We use the partition-tree data structure of Matoušek [3, 11, 13]. It supports half-space queries on sets with n points in time $O(\sqrt{n})$, linear space, and preprocessing time $O(n \log n)$.

We build a partition tree $\mathcal{T}(S)$ on the set S of points dual to the lines in S^* (in fact, these are the points to which a line is being fitted). A standard construction proceeds as follows: With each node v of the partition tree we associate a point set $S(v) \subseteq S$ and a triangle $\Delta(v) \supset S(v)$, where $S(v) \subset S(\text{parent}(v))$ at any node v other than the root and $S(v) = S$ at the root. In addition we also store at v the size $|S(v)|$ of $S(v)$ and the sums $\xi(S(v)) = \sum_{p_i \in S(v)} x_i$ and $\chi(S(v)) = \sum_{p_i \in S(v)} y_i$ of the slopes and constant terms of the corresponding dual lines. Since the point sets $S(v)$, over all children v of a node w in the tree, by definition of a partition tree, partition the set $S(w)$, and $|S(v)|$ is at most a fraction of $|S(w)|$, this tree has linear size and logarithmic depth. For our purposes, we modify the partition tree to obtain a new tree $\mathcal{T}(S, P)$ where the same $\Delta(v)$ as in $\mathcal{T}(S)$ is associated with every node v , but v stores $P(v) = S(v) \cap P$, $\xi(P(v))$ and $\chi(P(v))$ instead of the corresponding values for $S(v)$. This data structure enables us to execute the half-plane range query in P , and thus the vertical ray query in P^* .

Our data structure is semi-dynamic. When P changes, with a point p being added or removed, what we need to update is just values $|P(v)|$, $\xi(P(v))$, and $\chi(P(v))$ for each node v where p is relevant. Since the sets $S(v)$ for all nodes v at a fixed level of the partition tree form a partition of S , only one node must be updated at each level; to facilitate the update one might associate with each point $p \in S$ a list of length $O(\log n)$ containing the nodes v of the tree with $p \in S(v)$. Thus, the update can be performed in $O(\log n)$ time. This ends the description of the semi-dynamic vertical ray query data structure. Our sum-queries can be done by using the vertical ray query.

We next turn to the median-location query data structure. For a given t , let $m(t) = (t, y(t))$ be the intersection of the vertical line $X = t$ and the median level of the dual arrangement $\mathcal{A}(S^*)$. We can use the vertical query data structure to compare any given η with $y(t)$. We perform a vertical ray query to find the number of lines above (t, η) . If it is less than $\lfloor n/2 \rfloor$, $y(t) < \eta$; otherwise $y(t) \geq \eta$. This suggests computing $y(t)$ by some kind of binary search. If we had the sorted list of intersections between the vertical line $X = t$ and the lines in S^* available, we could perform a binary search on L by using $O(\log n)$ ray queries. However, it takes $O(n \log n)$ time to compute the list, which is too expensive since we aim for a sublinear query time. Instead, we construct a data structure which can simulate the binary search without explicitly computing the sorted list.

Lemma 6. *We can construct a randomized data structure in time $O(n \log n)$ such that, given t , we can compute $y(t)$ in $\tilde{O}(\sqrt{n})$ time. The query time bound holds for every vertical line $X = t$ with high probability.*

Proof. We fix a small constant $\varepsilon > 0$, and randomly select $cn^{1-\varepsilon}$ lines from $\Psi_0 = S^*$, to have a set Ψ_1 of lines, where c is a suitable constant. From the results of Clarkson and Shor [5], if the constant c is sufficiently large, with high probability every vertical segment intersecting no line of Ψ_1 intersects at most $n^\varepsilon \log n$ lines of S^* . In other words, Ψ_1 is the dual of an $(n^{\varepsilon-1} \log n)$ -net of S . Similarly, we construct Ψ_{i+1} from Ψ_i such that Ψ_{i+1}^* is an $(\frac{n^\varepsilon \log n}{|\Psi_i|})$ -net of Ψ_i^* if $|\Psi_i| > n^\varepsilon \log n$. Thus, we have a filtration $\Psi_0 \supset \Psi_1 \supset \dots \supset \Psi_k$,

and $|\Psi_k| \leq n^\varepsilon$. The number k of layers is a function of ε and c only, so the construction takes $O(n)$ time.

Additionally, we construct a dual range-searching data structure for Ψ_i such that for a query vertical interval I we can report all lines in Ψ_i meeting I in $O(\sqrt{n} + K)$ time, where K is the number of reported lines. In primal space a vertical interval corresponds to a strip bounded by two parallel lines and hence we may use partition trees as described above to implement reporting queries. The preprocessing time is $O(n \log n)$.

Now, our algorithm for finding $y(t)$ is as follows: Given t , we first compute all the intersections between $X = t$ and the lines of Ψ_k , sort them, and perform binary search for $y(t)$ on them. Each step of the search requires a vertical ray query and hence time $O(\sqrt{n})$. As the result of the binary search, we obtain a vertical interval I containing $y(t)$ such that no line of Ψ_k crosses the interior of I . By using the dual range-searching data structure, we extract, in time $O(\sqrt{n} + K)$, the set of K lines in Ψ_{k-1} intersecting I ; $K = O(n^\varepsilon \log n)$ with high probability. Proceeding recursively, we obtain $y(t)$, since at the last level of the filtration we arrive at an interval I' containing $y(t)$, with no line of $\Psi_0 = S^*$ crossing its interior. The total time is $O(n^\varepsilon \log^2 n + \sqrt{n} \log n) = \tilde{O}(\sqrt{n})$.

At this point, we have a $\tilde{O}(\sqrt{n})$ realization of the semi-dynamic query data structure, i.e., $\tau(n) = \sqrt{n}$. We finally come to the strongly polynomial method for determining a_{opt} . We use parametric search [16]. We use a parallel version of the ray-query algorithm, i.e., the parallel traversal of the partition tree, for the guide algorithm (see [9]). Since the depth of a partition tree is $O(\log n)$, the parallel time complexity of the ray query is $O(\log n)$. Thus, the parallel time complexity of sum queries is $O(\log^2 n)$ using $O(\tau(n))$ processors. Therefore, using standard parametric search paradigm, we can compute the optimal L_1 linear fitting in $\tilde{O}(\tau(n))$.

We remark that we do not employ parametric search to compute $y(t)$ for a fixed t , since it is not always possible to use it in a nested fashion, and there are technical difficulties in applying multi-dimensional parametric search paradigm [14] to our problem.

To speed up the query time $\tau(n)$ and thus the overall algorithm, we generalize the data structure to allow it to use super-linear storage based on Matoušek's construction [12]. If we can use $O(m)$ space for $n < m < n^2$, we first select $r = O(m/n)$ points from S and construct a dual cutting, i.e., a decomposition of the dual plane into cells, such that each cell C is intersected by at most n/r lines dual to points of S ; the number of cells required is $O(r^2)$ and the computation time is $O(nr)$. Let $S(C)$ be the set of lines intersecting C . We construct a point-location data structure on the cutting. For each cell C , we store the cumulative statistics (the sum of slopes etc.) for the set of lines passing below C , and construct the partition tree for $S(C)$. The query time of each tree is $\tilde{O}(\sqrt{n/r})$. When P changes, we need to update the data stored in each of the $O(r^2)$ cells of the arrangement, and also the $O(r)$ partition trees corresponding to sets containing the updated point. Thus, update time is $O(r^2 + r \log n)$. Update time can be sped up by not storing the statistics for each cell explicitly, but rather retrieving them when needed at a cost of $O(\log r)$. This reduces the time needed for an update to $O(r \log n)$ and the total time of all updates to $O(nr \log n)$; we omit the details in this version. If we set $r = n^{1/3}$, the update time and query time are both $\tilde{O}(n^{1/3})$. The space and preprocessing time is $\tilde{O}(n^{4/3})$. The parallel time complexity is not affected by the space-time trade-off.

Algorithm for L_1 1-Joint Fitting. Finally, we describe the algorithm to find the L_1 -optimal 1-joint polyline fitting a set S of n points in the plane. Recall that there are two different types of solutions:

Type 1. There is an index q such that the 1-joint polyline consists of the optimal L_1 -fitting line of $S_1(q) = \{p_1, p_2, \dots, p_q\}$ and that of $S_2(q) = \{p_{q+1}, p_{q+2}, \dots, p_n\}$.

Type 2. There is an index q such that the joint lies on the vertical line $x = x_q$.

If the optimal solution is of type 1, we compute an optimal L_1 -fitting line for $S_1(q)$ and $S_2(q)$ separately, for every $q = 1, 2, \dots, n$, by using the semi-dynamic algorithm with S as the universe. If we use quasi-linear space $\tilde{O}(n)$, the time complexity is $\tilde{O}(n^{1.5})$, and if we use $O(n^{4/3})$ space, the time complexity is $\tilde{O}(n^{4/3})$.

Otherwise, the optimal solution is of type 2. For each q , we guess the y -coordinate value η of the joint vertex (x_q, η) . Then, we can compute the best line, in the sense of L_1 fitting, approximating $S_1(q)$ going through the (for now, fixed) joint by using almost the same strategy as in section 3.2. Indeed, it suffices to determine the slope of this line. In the dual space, we just need to compute a point $p = (a(p), b(p))$ on the line $Y = x_q X - \eta$ such that $\sum_{i=1}^q |a(p)x_i - b(p) - y_i|$ is minimized. We observe that the above function is convex if it is regarded as a function of a , and hence $\theta(p) = \theta^+(p) - \theta^-(p)$ is monotone and changes the sign at p , where $\theta^+(p)$ ($\theta^-(p)$) is the sum of slopes of lines above p (resp. below p). Thus, we can apply binary search by using slope-sum query, and this binary search can be performed in $O(\log n)$ steps by using the filtration as described in Lemma 6.

Moreover, because of the convexity of the objective function, once we know the optimal solution for a given η_0 , we can determine whether the global optimal value η is greater than η_0 or not by using the height-sum query. Indeed, when we infinitesimally slide η_0 , the gain (or loss) of the objective function can be computed from the slope sums and height sums of dual lines associated with each of the sets of points lying above, below, and on the current polyline (for each of $S_1(q)$ and $S_2(q)$).

Thus, we can apply binary search for computing the optimal value of η . In order to construct a strongly polynomial algorithm, we apply parametric search. Note that given η , our algorithm has a natural parallel structure inherited from the range-searching algorithms, and runs in polylogarithmic time using $\tilde{O}(\tau(n))$ processors. Thus, the parallel search paradigm [16] is applicable here. Therefore, for a fixed q , the second case of the problem can be handled in $\tilde{O}(\tau(n))$ time. Thus, we have the following:

Theorem 2. *The optimal L_1 -fitting 1-joint polyline is computed in $\tilde{O}(n^{1.5})$ randomized time using quasi-linear space, and $\tilde{O}(n^{4/3})$ randomized time using $O(n^{4/3})$ space.*

4 Fitting a k -Joint Polyline

The k -joint fitting problem is polynomial-time solvable if k is a fixed constant. We describe the algorithm in a non-deterministic fashion. We guess the partition of x_1, \dots, x_n into k intervals each of which corresponds to a line segment in the polyline. Also, we guess whether each joint is free or fixed. We decompose the problem at the free joints

and have a set of subproblems. In each subproblem, we add the linear constraints corresponding to the fixed condition (i.e., each joint is located on a guessed vertical line). Thus, each subproblem is a convex programming problem: a linear program for L_1 , and a quadratic program for L_2 . We solve each subproblem separately to obtain the solution of the whole problem. Note that this strategy works because of the convexity of each subproblem. There are $O((3n)^k)$ different choices of the guesses, thus we can replace guessing by a brute-force search to have a polynomial-time deterministic algorithm if k is a constant.

For a general k , we do not know whether the problem is in the class P or not. Thus, we would like to consider approximation algorithms. One possible approach is to relax the requirement that number of joints be exactly k . We can design a PTAS for it.

Theorem 3. *Let z_{opt} be the optimal L_1 (or L_2) error of a k -joint fitting. Then, for any constant $\varepsilon > 0$, we can compute a $\lfloor(1 + \varepsilon)k\rfloor$ -joint fitting whose error is at most z_{opt} in polynomial time.*

Proof. We ignore continuity and approximate the points by using a piecewise-linear (not necessarily continuous) function with k linear pieces. This can be done by preparing the optimal linear regression for each subinterval of consecutive points of S , and then applying dynamic programming. We can restore the continuity by inserting at most k steep (nearly vertical) line segments. The resulting polyline has at most $2k$ joints and error at most z_{opt} . We can improve $2k$ to $\lfloor \frac{3k}{2} \rfloor$ by applying the 1-joint algorithm instead of linear regression algorithm, and further improve it to $\lfloor(1 + \varepsilon)k\rfloor$ by using the r -joint algorithm mentioned above for $r = \lceil \varepsilon^{-1} \rceil$.

Another approach is to keep the number of joints at k and approximate the fitting error. We give a FPTAS for it. We only discuss the L_1 case, since the L_2 case is analogous. Let z_{opt} be the optimal L_1 -error, and we aim to find a k -joint polyline whose error is at most $(1 + \varepsilon)z_{\text{opt}}$. We remark that if $z_{\text{opt}} = 0$, our solution is exactly the same as the solution for the uniform metric fitting problem, and thus we may assume $z_{\text{opt}} > 0$. Recall that the uniform metric fitting problem can be solved in $O(n \log n)$ time [6]. The following is a trivial but crucial observation:

Lemma 7. *Let z_∞ be the optimal error for the uniform metric k -joint fitting problem. Then, $z_\infty \leq z_{\text{opt}} \leq nz_\infty$.*

Proof. The sum of the errors in the uniform-metric-optimal polyline is at most nz_∞ . Hence $nz_\infty \geq z_{\text{opt}}$. On the other hand, every k -joint polyline has a data point in S such that the vertical distance to the polyline is at least z_∞ , so $z_{\text{opt}} \geq z_\infty$.

Our strategy is as follows: We call the n vertical lines through our input points the *column lines*. We give a set of *portal points* on each column line, and call a k -joint polyline a *tame polyline* if each of its links satisfies the condition that the line containing the link goes through a pair of portal points.

On each column line, the distance between its data point and the intersection point with the optimal polyline is at most z_{opt} , thus at most nz_∞ . Thus, on the i th column line, we place the portals in the vertical range $[y_i - nz_\infty, y_i + nz_\infty]$. The portal points are placed symmetrically above and below y_i . The j th portal above y_i is located at the

y -value $y_i + (1 + \frac{\varepsilon}{2})^{j-1} \delta$, where $\delta = \frac{z_\infty \varepsilon}{2n}$ and $j = 1, 2, \dots, M$. We choose the largest M satisfying $(1 + \frac{\varepsilon}{2})^M \delta \leq n z_\infty$, and hence $M = O(\varepsilon^{-1} \log(n + \varepsilon^{-1}))$. We also put portals at heights y_i and $y_i \pm n z_\infty$. In this way the number of portals in any column is at most $2M + 3$.

Lemma 8. *There exists a tame polyline whose L_1 error is at most $(1 + \varepsilon)z_{\text{opt}}$.*

We omit the proof of above lemma in this version. Thus, it suffices to compute the optimal tame polyline. There are Mn portals, and thus $N = O(M^2 n^2)$ lines going through a pair of portals. Let \mathcal{L} be the set of these lines. We design a dynamic programming algorithm. For the i th column, for each line $\ell \in \mathcal{L}$ and each $m \leq k$, we record the approximation error of the best m -joint tame polyline up to the current column whose (rightmost) link covering p_i is on ℓ . When we proceed to the $(i + 1)$ th column, each approximation error is updated. We omit the details of the analysis in this version, and only show the result.

Theorem 4. *An $(1 + \varepsilon)$ -approximation, i.e., a k -joint polyline with error $1 + \varepsilon$ times the optimal, for each of the L_1 and L_2 k -joint problems can be computed in $O(kn^4 \varepsilon^{-4} \log^4(n + \varepsilon^{-1}))$ time.*

5 Concluding Remarks

A major open problem is to determine the complexity class of the k -joint problem for L_1 - and L_2 -fitting. The corresponding L_1 or L_2 polyline approximation problem where the input is a curve is also interesting.

Acknowledgment: The authors would like to thank Jirí Matoušek for a stimulating discussion on convexity.

References

1. A. Aggarwal, B. Schieber, and T. Tokuyama, "Finding a minimum-weight k -link path in graphs with the concave Monge property and applications," *Discrete Comput. Geom.*, **12** (1994) 263–280.
2. P. Agarwal, B. Aronov, T. Chan, M. Sharir, "On Levels in Arrangements of Lines, Segments, Planes, and Triangles," *Discrete Comput. Geom.*, **19** (1998) 315–331.
3. P. Agarwal and J. Matoušek, "Ray shooting and parametric search," *SIAM J. Comput.*, **22** (1993) 794–806.
4. J. Chun, K. Sadakane, and T. Tokuyama, "Linear time algorithm for approximating a curve by a single-peaked curve," *Proc. 14th Internat. Symp. Algorithms Comput. (ISAAC 2003)*, LNCS 2906, 2003, pp. 6–16.
5. K. L. Clarkson and P. W. Shor, "Application of Random Sampling in Computational Geometry," *Discrete Comput. Geom.*, **4** (1989) 423–432.
6. M. Goodrich, "Efficient piecewise-linear function approximation using the uniform metric," *Discrete Comput. Geom.*, **14** (1995) 445–462.

7. S. Hakimi and E. Schmeichel, "Fitting polygonal functions to a set of points in the plane," *Graphical Models and Image Processing*, **53** (1991) 132–136.
8. H. Imai, K. Kato, and P. Yamamoto: "A linear-time algorithm for linear L_1 approximation of points," *Algorithmica*, **4** (1989) 77–96.
9. N. Katoh and T. Tokuyama, "Notes on computing peaks in k-levels and parametric spanning trees," Proc. 17th ACM Symp. on Computational Geometry, 2001, pp. 241–248.
10. S. Langerman and W. Steiger, "Optimization in arrangements." *Proc. Symp. Theor. Aspects Computer Science (STACS2003)*, LNCS 2607, 2003, pp. 50–61.
11. J. Matoušek, "Efficient partition trees," *Discrete Comput. Geom.*, **8** (1992) 315–334.
12. J. Matoušek, "Range searching with efficient hierarchical cutting," *Proc. 8th ACM Symp. on Comput. Geom.*, (1992) 276–287.
13. J. Matoušek, "Geometric range searching," *ACM Computing Surveys*, **26** (1994) 421–461.
14. J. Matoušek and O. Schwarzkopf, "Linear optimization queries," *Proc. 8th Annual ACM Symp. Comput. Geom.*, 1992, pp. 16–25.
15. F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, New York, 1985.
16. J. Salowe, "Parametric Search," Chapter 37 of *Handbook of Discrete and Computational Geometry* (eds. J. Goodman and J. O'Rourke), CRC Press, 1997.

A Generalization of Magic Squares with Applications to Digital Halftoning

Boris Aronov^{1,*}, Tetsuo Asano^{2,**}, Yosuke Kikuchi³, Subhas C. Nandy⁴,
Shinji Sasahara⁵, and Takeaki Uno⁶

¹ Polytechnic University, Brooklyn, NY 11201-3840, USA

<http://cis.poly.edu/~aronov>

² JAIST, Tatsunokuchi, 923-1292 Japan

t-asano@jaist.ac.jp

³ ERATO QCI Project, JST, Tokyo 113-0033, Japan

kikuchi@qci.jst.go.jp

⁴ Indian Statistical Institute, Kolkata 700 108, India

nandysc@isical.ac.in

⁵ Fuji Xerox Co., Ltd., Kanagawa 259-0157, Japan

shinji.sasahara@fujixerox.co.jp

⁶ National Institute of Informatics (NII), Tokyo, 101-8430 Japan

uno@nii.jp

Abstract. A *semimagic square* of order n is an $n \times n$ matrix containing the integers $0, \dots, n^2 - 1$ arranged in such a way that each row and column add up to the same value. We generalize this notion to that of a *zero $k \times k$ -discrepancy matrix* by replacing the requirement that the sum of each row and each column be the same by that of requiring that the sum of the entries in each $k \times k$ square contiguous submatrix be the same. We show that such matrices exist if k and n are both even, and do not if k and n are relatively prime. Further, the existence is also guaranteed whenever $n = k^m$, for some integers $k, m \geq 2$. We present a space-efficient algorithm for constructing such a matrix.

Another class that we call *constant-gap matrices* arises in this construction. We give a characterization of such matrices.

An application to digital halftoning is also mentioned.

1 Introduction

A *semimagic square* is an $n \times n$ matrix filled with the numbers $0, \dots, n^2 - 1$ in such a way that the sum of the numbers in each row and each column are the same. Magic squares and related classes of integer matrices have been studied extensively (for an exhaustive bibliography, see [8] and the references therein).

This paper generalizes the notion of a semimagic square by replacing the requirement that all row and column sums be the same by the analogous requirement for all $k \times k$ contiguous square submatrices; we call such $n \times n$ matrices *zero $k \times k$ -discrepancy matrices* of order (k, n) . Let $\mathbb{N}(k, n)$ be the set of all such matrices. In this paper we prove

* Part of the work on the paper has been carried out when B.A. was visiting JAIST. Work of B.A. on this paper was supported in part by NSF ITR Grant CCR-00-81964.

** Work of T.A. was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B).

that $\mathbb{N}(k, n)$ is non-empty if k and n are both even, and empty if they are relatively prime. Further, we show by an explicit construction that $\mathbb{N}(k, k^m) \neq \emptyset$ for any integers $k, m \geq 2$.

Another property plays an important role in the latter construction of zero $k \times k$ -discrepancy matrices. A characterization of matrices with this property is also given in this paper.

Our investigation is motivated by an application described below, but intuitively we seek a matrix filled with distinct integers in an as uniform a manner as possible. The analogous geometric problem of distributing n points uniformly in a unit square has been studied extensively in the literature [6, 12]. Usually, a family of regions is introduced to evaluate the uniformity of a point distribution. If the points of an n -point set P are uniformly distributed, for any region R in the family the number of points in R should be close to $\frac{1}{n} \text{area}(R)$, where $\frac{1}{n}$ is the point density of P in the entire square. Thus, the *discrepancy of P in a region R* is defined as the difference between this value and the actual number of points of P in R . The *discrepancy of the point distribution P* with respect to the family of regions is defined by the maximum such difference, over all regions.

The problem of establishing discrepancy bounds for various classes of regions has been studied extensively [10]. One of the simplest families is that of axis-parallel rectangles for which $\Theta(\log n)$ bound is known [6, 12]. In the context of digital halftoning, a family of axis-parallel squares (contiguous square submatrices) over a matrix is appropriate for measuring the uniformity since human eye perception is usually modeled using weighted sum of intensity levels with Gaussian coefficients over square regions around each pixel [3]. Thus, the matrices discussed in this paper can be used as dither matrices in which integers are arranged in an apparently random manner to be used as variable thresholds. Small matrix size tends to generate visible artifacts. In this sense the dither matrix of size 8×8 designed by Floyd and Steinberg [7] may be too small. A common way to construct a larger dither matrix is to use local search under some criterion based on spatial frequency distribution of the resulting matrix. Such dither matrices are called blue-noise masks [13, 15, 16, 17]. One disadvantage of a blue-noise mask is its high space complexity. There appears to be no way to avoid storing the entire matrix. The zero $k \times k$ -discrepancy matrices of order (k, k^m) we construct, on the other hand, are such that we can generate any one element by a simple integer calculation requiring only m seed matrices, each of size $k \times k$.

2 Problem Statement

Generalizing the notion of a semimagic square, we consider an $n \times n$ matrix containing all the integers $0, \dots, n^2 - 1$ such that the entries contained in every contiguous $k \times k$ submatrix add up to the same value.

More formally, for integers $m, n > 1$, let $\mathbb{Z}(n, m)$ be the class of all $n \times n$ integer matrices with entries from the set $\{0, \dots, m - 1\}$ and let $\mathbb{Z}(n) \subset \mathbb{Z}(n, n^2)$ be the set of those $n \times n$ matrices which contain every value $0, \dots, n^2 - 1$ exactly once.

A contiguous $k \times k$ submatrix (or *region*, hereafter) $R_{i,j} = R_{i,j}^{(k)}$ with its upper left corner at (i, j) is defined by

$$R_{i,j}^{(k)} = \{(i', j') \mid i' = i, \dots, i + k - 1 \text{ and } j' = j, \dots, j + k - 1\},$$

where indices are calculated modulo n .¹ Given a matrix P and a region $R_{i,j}$ of size k , $P(R_{i,j})$ denotes the sum of the elements of P in locations given by $R_{i,j}$. Analogously, define a $C_{i,j} = C_{i,j}^{(k)}$ to be the $k \times 1$ region of a matrix starting at (i,j) and $P(C_{i,j})$ to be the sum of elements of P in the locations given by $C_{i,j}$. We are interested in *all* $k \times k$ regions in an $n \times n$ matrix:

$$\mathcal{F}_{k,n} = \{R_{i,j}^{(k)} \mid i, j = 0, 1, \dots, n-1\}.$$

The $k \times k$ -discrepancy $\mathcal{D}_{k,n}(P)$ of an $n \times n$ matrix P for the family $\mathcal{F}_{k,n}$ is defined as

$$\mathcal{D}_{k,n}(P) = \max_{R \in \mathcal{F}_{k,n}} P(R) - \min_{R' \in \mathcal{F}_{k,n}} P(R').$$

In this paper we focus on the existence of matrices $P \in \mathbb{Z}(n)$ with $k \times k$ -discrepancy $\mathcal{D}_{k,n}(P) = 0$. In other words, we are interested in the existence and construction of matrices in $\mathbb{Z}(n)$ all of whose contiguous $k \times k$ submatrices have equal sums. Let $\mathbb{N}(k,n)$ be the set of all such zero- $k \times k$ -discrepancy matrices of order (k,n) .

Theorem 1. *The set $\mathbb{N}(k,n)$ of zero- $k \times k$ -discrepancy matrices of order (k,n) has the following properties:*

- (a) $\mathbb{N}(k,n)$ is non-empty if k and n are both even.
- (b) $\mathbb{N}(k,n)$ is empty if k and n are relatively prime.
- (c) $\mathbb{N}(k,n)$ is empty if k is odd and n is even.
- (d) $\mathbb{N}(k,k^m)$ is non-empty for any integers k and m , $k \geq 2, m \geq 2$.

Proof (Theorem 1, parts (a)–(c)). To prove part (a), it suffices to show $\mathbb{N}(2,n) \neq \emptyset$ if n is even since any $k \times k$ region can be partitioned into 2×2 regions if k is even. (More generally, if k' divides k , $\mathbb{N}(k',n) \subset \mathbb{N}(k,n)$.)

Let $P = (p_{i,j}) \in \mathbb{Z}(n)$ be the matrix in which the numbers are arranged in the row-major order, that is, $p_{i,j} = in + j, i, j = 0, 1, \dots, n-1$. We classify matrix elements by their parity and rotate all the elements of odd parity by 180 degrees, i.e., for every (i,j) with $i+j$ odd, we swap $p_{i,j}$ and $p_{n-1-i, n-1-j}$. It is easily checked that the sum of elements in any 2×2 region is always $2n^2 - 2$. An example for $n = 8$ is shown in Fig. 1.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 62 & 2 & 60 & 4 & 58 & 6 & 56 \\ 55 & 9 & 53 & 11 & 51 & 13 & 49 & 15 \\ 16 & 46 & 18 & 44 & 20 & 42 & 22 & 40 \\ 39 & 25 & 37 & 27 & 35 & 29 & 33 & 31 \\ 32 & 30 & 34 & 28 & 36 & 26 & 38 & 24 \\ 23 & 41 & 21 & 43 & 19 & 45 & 17 & 47 \\ 48 & 14 & 50 & 12 & 52 & 10 & 54 & 8 \\ 7 & 57 & 5 & 59 & 3 & 61 & 1 & 63 \end{bmatrix}$$

Fig. 1. Parity rotation used in the proof of Theorem 1(a)

Turning to part (b), for a contradiction, assume that there exists a matrix $P \in \mathbb{Z}(n)$ in which the sum $P(R_{i,j})$ of elements of P over a $k \times k$ region $R_{i,j}$ is independent of i, j .

¹ Throughout this paper, index arithmetic is performed modulo matrix dimensions unless otherwise noted.

In particular, $P(R_{i,j}) = P(R_{i,j+1}) = c$ for some constant c and therefore $P(C_{i,j}) = P(C_{i,j+k}) = c/k$, for all i, j .

Since k and n are relatively prime, the last relation implies that in fact $P(C_{i,j})$ is independent of j . Similar reasoning leads to the conclusion that it is independent of i as well. In particular, $P(C_{0,0}) = P(C_{1,0})$, and therefore, by definition of $C_{0,0}$ and $C_{1,0}$, we must have $p_{i,0} = p_{i+k,0}$, contradicting our assumption that all the elements of P are distinct.

Finally, we consider part (c) of Theorem 1. Let $P \in \mathbb{Z}(n)$ and let k be odd and n be even. For a contradiction, assume that the values in any $k \times k$ region add up to the same number, say S , which must clearly be an integer. Summing $P(R_{i,j})$ over all i and j and observing that every entry in P appears precisely k^2 times in these sums, we conclude that

$$n^2 S = k^2(0 + 1 + \dots + n^2 - 1) = k^2 \frac{n^2}{2}(n^2 - 1),$$

and therefore $S = k^2(n^2 - 1)/2$, which cannot be an integer if n is even and k is odd. This contradiction concludes the proof of Theorem 1(a)–(c). □

3 Construction of a $k^m \times k^m$ -Matrix of Zero $k \times k$ -Discrepancy

In this section we finish the proof of Theorem 1 by designing a $k^m \times k^m$ matrix from $\mathbb{Z}(k^m)$ for any positive integer m such that its $k \times k$ -discrepancy is zero; in fact we present a proof of a stronger statement, see Theorem 3. We first show that there exists a $k^2 \times k^2$ matrix in $\mathbb{Z}(k^2)$ whose $k \times k$ discrepancy is zero, and then extend the result to $k^m \times k^m$ matrices.

Definition 1. The simple expansion \tilde{P} of a $k \times k$ matrix P is the matrix formed by repeating P $k \times k$ times, as follows:

$$\tilde{P} = \begin{bmatrix} P & P & \dots & P \\ P & P & \dots & P \\ & & \ddots & \\ P & P & \dots & P \end{bmatrix}.$$

Note that the $k \times k$ -discrepancy of \tilde{P} is zero, as every $k \times k$ region contains the same set of numbers.

Definition 2. A cyclic column shift of a matrix P is the matrix obtained by shifting each column of P to the right (i.e., shifting the j th column to the $(j + 1)$ st column) and moving the last column to the first column. A cyclic row shift is similarly defined: it means shifting each row of P down to the next lower row (i.e., shifting i th row to the $(i + 1)$ st row) and moving the bottom row to the top row.

We denote the matrix obtained by applying cyclic column shift c times and cyclic row shift r times to a $k \times k$ matrix P by $P^{(c,r)}$. That is, element (i, j) in P moves to position $((i + r) \bmod k, (j + c) \bmod k)$ in $P^{(c,r)}$. The cyclic expansion $\hat{P} = (\hat{p}_{i,j})$ of a $k \times k$ matrix P is a $k^2 \times k^2$ matrix defined by

$$\hat{P} = \begin{bmatrix} P^{(0,0)} & P^{(0,1)} & \dots & P^{(0,k-1)} \\ P^{(1,0)} & P^{(1,1)} & \dots & P^{(1,k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ P^{(k-1,0)} & P^{(k-1,1)} & \dots & P^{(k-1,k-1)} \end{bmatrix}.$$

An easy calculation shows that, for all i, j , $\hat{p}_{i,j} = p_{i',j'}$, with

$$i' = i + \lfloor j/k \rfloor \text{ and } j' = j + \lfloor i/k \rfloor \pmod{k}. \tag{1}$$

Definition 3. A constant-gap matrix $P = (p_{i,j})$ is one for which

$$p_{i,j} - p_{i,j'} = p_{i',j} - p_{i',j'} \tag{2}$$

holds for all choices of i, i', j , and j' .

Intuitively, this means that for any two columns j and j' the gap between elements in the same row is independent of the row, hence the ‘‘constant gap’’ name. Since (2) can be rewritten as

$$p_{i,j} - p_{i',j} = p_{i,j'} - p_{i',j'} \text{ or } p_{i,j} + p_{i',j'} = p_{i,j'} + p_{i',j},$$

rows and columns play symmetric roles in the definition. Moreover, a constant-gap matrix has the strong Monge property [1] since the sum of the main diagonal elements is equal to that of the off diagonal elements in any 2×2 submatrix.

Lemma 1. *The constant-gap property is preserved (1) under exchange of any two rows, (2) under exchange of any two columns, and, for square matrices, (3) under mirror reflection across the main diagonal.*

Proof. Immediate from the definition. □

The following lemma is a key to our construction of zero discrepancy matrices.

Lemma 2. *If P is a $k \times k$ constant-gap matrix, the $k \times k$ -discrepancy of its cyclic expansion \hat{P} is zero.*

Proof. Recall that $R_{i,j}$ and $C_{i,j}$ denote $k \times k$ and $k \times 1$ contiguous submatrices of \hat{P} , and $\hat{P}(R_{i,j})$ and $\hat{P}(C_{i,j})$ the sums of the corresponding elements in \hat{P} , respectively. We aim to prove $\hat{P}(R_{i,j}) = \hat{P}(R_{i,j+1})$, for all i, j . Together with $\hat{P}(R_{i,j}) = \hat{P}(R_{i+1,j})$, which is proven by a symmetric argument, this implies the statement of the theorem. By definition, $\hat{P}(R_{i,j+1}) - \hat{P}(R_{i,j}) = \hat{P}(C_{i,j+k}) - \hat{P}(C_{i,j})$; recall that all indices in \hat{P} are calculated modulo k^2 .

Put $i_0 = k \lfloor i/k \rfloor$ and $j_0 = k \lfloor j/k \rfloor$. To prove $\hat{P}(C_{i,j}) = \hat{P}(C_{i,j+k})$ we compare the two columns. As illustrated in Fig. 2, the part above the element $(i_0 + k - 1, j)$ and the one above the element $(i + k - 1, j)$ in $C_{i,j}$ both appear in $C_{i,j+k}$. Differences between $C_{i,j}$ and $C_{i,j+k}$ comprise only four elements: $a = \hat{p}_{i_0+k-1,j}$, $b = \hat{p}_{i+k-1,j}$, $c = \hat{p}_{i,j+k}$, $d = \hat{p}_{i_0+k,j+k}$. By cyclic row and column shifts, the four elements move in \hat{P} as follows:

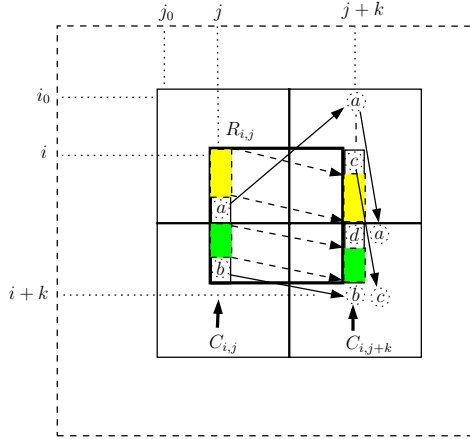


Fig. 2. Illustration to the proof of Lemma 2

$$\begin{aligned}
 a &= \hat{p}_{i_0+k-1,j} \rightarrow_r \hat{p}_{i_0,j+k} \rightarrow_c \hat{p}_{i_0+k,j+k+1}, \\
 b &= \hat{p}_{i+k-1,j} \rightarrow_r \hat{p}_{i+k,j+k}, \\
 c &= \hat{p}_{i,j+k} \rightarrow_c \hat{p}_{i+k,j+k+1}, \\
 d &= \hat{p}_{i_0+k,j+k},
 \end{aligned}$$

where \rightarrow_x represents the cyclic x shift and indices are calculated modulo k^2 .

When $j \neq k-1 \pmod k$ and $i \neq k-1 \pmod k$, all four elements $d: \hat{p}_{i_0+k,j+k}$, $a: \hat{p}_{i_0+k,j+k+1}$, $b: \hat{p}_{i+k,j+k}$, and $c: \hat{p}_{i+k,j+k+1}$ belong to the same submatrix, namely, to $P(\lfloor i_0/k \rfloor + 1, \lfloor j_0/k \rfloor + 1)$. Since the constant gap property is preserved by cyclic row and column shifts, we have $d-b = a-c$, and thus $a+b = c+d$. The cases when $j \neq k-1 \pmod k$ and/or $i \neq k-1 \pmod k$ are handled by a variant of this argument; we omit the details in this version. This completes the proof of $\hat{P}(C_{i,j}) = \hat{P}(C_{i,j+k})$ and of the lemma. \square

Lemma 3. Let $P = (p_{ij})$ and $Q = (q_{ij})$ be matrices in $\mathbb{Z}(k)$. Combine \hat{P} and \tilde{Q} into a single matrix in two different ways, namely, put $C^{(1)} = C^{(1)}(P, Q) = (c_{ij}) = \tilde{Q} + k^2 \hat{P}$ and $C^{(2)} = C^{(2)}(P, Q) = (c'_{ij}) = \hat{P} + k^2 \tilde{Q}$. In other words, $c_{i,j} = \tilde{q}_{i,j} + k^2 \hat{p}_{i,j}$ or $c'_{i,j} = \hat{p}_{i,j} + k^2 \tilde{q}_{i,j}$, for all i, j . If P has the constant gap property, then

- (a) $C^{(1)}$ and $C^{(2)}$ are in $\mathbb{Z}(k^2)$, and
- (b) their $k \times k$ -discrepancy is zero.

In addition, $C^{(1)}$ and $C^{(2)}$ are distinct if $P \neq Q$. Thus $|\mathbb{N}(k, k^2)| \geq 2$.

Proof. The resulting matrices obviously belong to $\mathbb{Z}(k^2, k^4)$ and have zero discrepancy, as linear combinations of matrices of zero discrepancy. It is easy to check that $C^{(1)} \neq C^{(2)}$ if $P \neq Q$.

Thus to prove (b), it suffices to show that the elements of the matrices are all distinct. We focus on $C^{(1)}$, the argument for $C^{(2)}$ is analogous. Since $\hat{P}, \tilde{Q} \in \mathbb{Z}(k^2, k^2)$, $c_{ij} = c_{i'j'}$

The remainder of the proof proceeds just as in that of Lemma 3; we omit the details. Recall that $P_m^{(a,b)}(i,j) = P_m((i+b) \bmod k, (j+a) \bmod k)$. Thus we can generate every entry of such a matrix without explicitly storing any information besides the m $k \times k$ matrices P_0, \dots, P_{m-1} ; the computation requires at most $O(m)$ additional working space. \square

4 The Class of Constant-Gap Matrices

We have described a scheme for constructing matrices with zero $k \times k$ -discrepancy. A key ingredient in the recipe is a constant-gap matrix in $\mathbb{Z}(k)$. It is easily checked that a different choice of such a matrix produces a different zero-discrepancy matrix. Thus a natural question arises: How many different constant-gap matrices of a given size are there? In this section we, in a sense, completely characterize the class of constant-gap matrices in $\mathbb{Z}(n)$. In fact, we discuss a somewhat more general class of matrices. Let $\mathbb{M}(m,n)$ be the set of all integer $m \times n$ matrices with entries $0, \dots, mn-1$, each used exactly once. A matrix $M = (m_{ij}) \in \mathbb{M}(m,n)$ has *constant-gap property* if, for all i, j, i', j' , $m_{ij} + m_{i'j'} = m_{i'j} + m_{ij'}$. It is clear from the definition that this property, as already observed in Lemma 1, is invariant under a number of operations:

Lemma 4. *The constant-gap property is preserved (1) under an arbitrary permutation of rows of a matrix, (2) under an arbitrary permutation of columns of a matrix, and, for square matrices, (3) under mirror reflection through its main diagonal.*

Two constant-gap matrices are *equivalent* if one of them is derived from the other by a sequence of operations listed in the statement of the lemma.² We are interested in counting the number of these equivalence classes.

Lemma 5. *Every equivalence class can be represented by a matrix $P = (p_{i,j}) \in \mathbb{M}(m,n)$ in canonical form, which satisfies the following additional properties:*

- (a) *Every row of P is sorted, i.e., $p_{i,0} < p_{i,1} < \dots < p_{i,n-1}$; for convenience we define $c_j = p_{0,j}$ for $j = 0, \dots, n-1$.*
- (b) *Every column of P is sorted, i.e., $p_{0,j} < p_{1,j} < \dots < p_{n-1,j}$; we put $r_j = p_{j,0}$, for $j = 0, \dots, n-1$.*
- (c) *Generally, $p_{i,j} < p_{i',j'}$ if $i \leq i', j \leq j'$, and $(i,j) \neq (i',j')$.*
- (d) *$p_{0,0} = c_0 = r_0 = 0$, $p_{m,n} = mn-1$.*
- (e) *P is completely specified by $(c_j), (r_i)$: for all i, j , $p_{i,j} = r_i + c_j$.*

Proof. We argue that all the properties can be satisfied without leaving the equivalence class. Columns can be permuted to sort the top row and then rows can be permuted to sort the leftmost column. Because of the constant-gap property, this sorts all rows and columns. Properties (c) and (d) follow from this ordering. The next property follows from the constant-gap condition, namely $p_{i,j} = p_{0,0} + p_{i,j} = p_{i,0} + p_{0,j} = r_i + c_j$. \square

² To avoid cumbersome wording, we will henceforth not discuss the case of square matrices separately. The reasoning there is entirely analogous, with the exception of an occasional invocation of diagonal symmetry to further reduce the number of equivalence classes. We omit further details.

$$\begin{array}{ccc}
 \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 4 & 5 \\ 2 & 3 & 6 & 7 \\ 8 & 9 & 12 & 13 \\ 10 & 11 & 14 & 15 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 8 & 9 \\ 2 & 3 & 10 & 11 \\ 4 & 5 & 12 & 13 \\ 6 & 7 & 14 & 15 \end{bmatrix} \\
 (a) & & (b) &
 \end{array}$$

Fig. 3. The only equivalence class of constant-gap matrices for $n = 3$ (a), and the three classes for $n = 4$ (b)

It is not difficult to see that there exists only one equivalence class (up to diagonal reflection) of constant-gap matrices in $\mathbb{Z}(n)$ when $n = 3$; for $n = 4$, there are three equivalence classes, refer to Fig. 3. For $n = 5$ there exists only one equivalence class, whereas there are many when $n = 6$. These observations can be generalized as follows.

The set of all constant-gap matrices in $\mathbb{M}(m, n)$ in canonical form is denoted by $\mathbb{K}(m, n)$. In this section we give a characterization of the sets $\mathbb{K}(m, n)$, for all $m, n > 0$. We begin with some additional terminology and then state our characterization.

We define another operation on matrices. Given matrices $P = (p_{i,j}) \in \mathbb{K}(m, n)$ and $Q = (q_{i,j}) \in \mathbb{K}(m', n')$, their *expansion product* $P \otimes Q$ is the $mm' \times nn'$ matrix $H = (h_{i,j})$ defined by $h_{i,j} = m'n'p_{\lfloor i/m' \rfloor, \lfloor j/n' \rfloor} + q_{i \bmod m', j \bmod n'}$. In addition, define a *simple row* of length k to be the $1 \times k$ matrix filled with consecutive numbers $0, \dots, k - 1$, in this order. Define a *simple column* analogously.

The following facts are easily verified.

Fact 1. $\mathbb{K}(1, m)$ consists of a single matrix, which is a simple row of length m . An analogous statement holds for $\mathbb{K}(m, 1)$. Both row- and column-major order filled members of $\mathbb{M}(m, n)$ have constant-gap property. In particular, $|\mathbb{K}(m, n)| \geq 2$, for $m, n > 1$. The two matrices have the same canonical form in $\mathbb{K}(n)$, so $|\mathbb{K}(n)| \geq 1$ for $n \geq 1$.

Fact 2. If $P \in \mathbb{K}(m, n)$ and $Q \in \mathbb{K}(m', n')$, then $P \otimes Q \in \mathbb{K}(mm', nn')$.

Surprisingly, in the sense made more precise by the following theorem, Facts 1 and 2 describe $\mathbb{K}(m, n)$ completely. The remainder of the section is devoted to the proof of this assertion.

Theorem 3. $\mathbb{K}(m, n)$ can be characterized as follows.

(a) A matrix $P \in \mathbb{K}(m, n)$ can be written uniquely as

$$P = P_1 \otimes \dots \otimes P_k,$$

where P_1, \dots, P_k is an alternating sequence of simple rows and columns, each of length at least two; $k = 0$ if $m = n = 1$.

(b) $\mathbb{K}(m, n) \neq \emptyset$, for $m, n > 0$. $|\mathbb{K}(m, n)| = 1$ if and only if $n = 1$ or $m = 1$. In this case, $\mathbb{K}(m, n)$ consists just a simple row or column.

A *column difference* $p_{0,j} - p_{0,k} = p_{i,j} - p_{i,k}$, for some $j \neq k$, is the difference between corresponding entries of two columns; it is independent of the row where the difference is taken, by the constant-gap property. Similarly, a *row difference* $p_{i,0} - p_{k,0} = p_{i,j} - p_{k,j}$, for $i \neq k$, is the difference between corresponding entries of two rows.

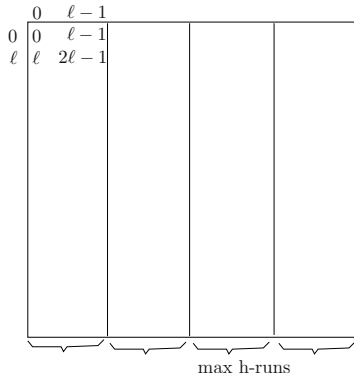


Fig. 4. Partition of a matrix by vertical boundaries of h-runs

Lemma 6. *No row difference equals a column difference. In other words, for any $0 \leq i < k \leq n - 1$ and $0 \leq j < \ell \leq n - 1$, $p_{0,k} - p_{0,i} \neq p_{\ell,0} - p_{j,0}$.*

Proof. By the constant-gap property, $p_{0,k} - p_{0,i} = p_{\ell,k} - p_{\ell,i}$ and $p_{\ell,0} - p_{j,0} = p_{\ell,k} - p_{j,k}$. Their equality would imply $p_{\ell,i} = p_{j,k}$ contradicting the assumption that no entry in P is repeated. \square

Proof (Theorem 3). We start with the existence proof for part (a). If $n = 1$ or $m = 1$, we are done, so assume $m, n > 1$. Without loss of generality, assume that $p_{0,1} < p_{1,0}$, so that $p_{0,1} = 1$. By induction, it is sufficient to argue that in this case P can be written as a product of a smaller matrix and a simple row of length at least two.

Let $P = (p_{i,j}) \in \mathbb{K}(m, n)$. A *horizontal run* in P (an *h-run*, for short) is a maximal sequence of consecutive integers appearing in adjacent entries of a row of P ; the *length* of an h-run is the number of such integers. Each h-run is associated with an interval defined by its first and last column indices. The *initial h-run* is the one starting at the upper left corner $p_{0,0}$ of the matrix; by our assumption its length ℓ is at least two. The value ℓ must be contained in $p_{0,1}$, for it cannot lie in $p_{0,\ell}$ by maximality of the run and it is the smallest value in the matrix outside of the run. Thus we have the row difference $p_{1,0} - p_{0,0} = \ell$.

Lemma 7. *No h-run is longer than the initial run.*

Proof. If there existed such an h-run, we would have a column difference equal to ℓ appear within the run. However, we already identified a row difference of ℓ in the matrix. This would contradict Lemma 6. \square

Recall that in P the differences between consecutive elements in the row are equal to the corresponding differences in the top row, so in terms of presence and length of runs, every row behaves exactly the same. Thus P can be partitioned by vertical lines corresponding to boundaries of h-runs, as shown in Fig. 4.

Lemma 8. *All h-runs have the same length.*

Proof. By Lemma 7, no h-run is longer than the initial h-run of length ℓ . For a contradiction, let j be the first column at which an h-run of length $\ell' < \ell$ starts. Such an h-run in the top row is a sequence $p_{0,j}, p_{0,j+1}, \dots, p_{0,j+\ell'} - 1$. Where in the matrix is the next integer, namely the value $q = p_{0,j} + \ell'$? By definition of an h-run, it cannot be located in the same row. So, it must lie before column j , say in location $p_{k,s\ell}$, for some $s\ell < j$. Since all the h-runs located to the left of column j are of length ℓ , the h-run starting at $p_{k,s\ell}$ must consist of values $q, \dots, q + \ell - 1$. However, since the difference between rows zero and one is ℓ , $p_{1,j}$ must have value $r = p_{0,j} + \ell > q$. Moreover, $r = q - \ell' + \ell < q + \ell - 1$, so the value r occurs both at $p_{1,j}$ and in the run starting at value q , which is impossible. \square

This property implies the following structure of the matrix P . The number $\ell > 1$ is a divisor of n . The entire matrix consists of h-runs of length ℓ . The first element of each run is a multiple of ℓ . If L is a simple row of length ℓ , we can easily verify that indeed $P = P' \otimes \ell$ for the matrix $P' = (p'_{i,j})$ defined by $p'_{i,j} = p_{i,\ell j} / \ell$. We claim that $P' \in \mathbb{K}(m, n/\ell)$. It is easily checked that $P' \in \mathbb{M}(m, n/\ell)$, while properties (a)–(e) follow from the corresponding properties for P .

We now address the question of uniqueness. We have shown that any matrix in $\mathbb{K}(m, n)$ can be written as an expansion product of some number of simple rows and columns. Since the expansion product of two simple rows (resp. columns) is a simple row (resp. column), by consolidating products of consecutive rows (resp. columns) we can express P as a product of alternating rows and columns. This representation is unique, since the type and size of the last factor can be “read” directly from the matrix—the factor is a simple row if $p_{0,1} = 1$ and a simple column if $p_{1,0} = 1$; its length ℓ is the length of the initial run, which is horizontal in the former case and vertical in the latter one.

It remains to note that a matrix from $\mathbb{K}(m, n)$ can always be produced as a product of a simple row of length n and a simple column of length m , thus constant-gap matrices of all orders exist. Reversing the order of multiplication produces a different matrix, provided $m, n > 1$, proving part (b) of the theorem. \square

5 Concluding Remarks

We have introduced a discrepancy-based measure of uniformity of an $n \times n$ square matrix containing $0, 1, \dots, n^2 - 1$ as a generalization of a semimagic square. We have succeeded in obtaining matrices of even dimension with zero discrepancy for families of $2k \times 2k$ contiguous submatrices. For arbitrary k , we can construct a $k^m \times k^m$ matrix of $k \times k$ -discrepancy zero. Moreover, such a matrix can be explicitly computed in time linear in its size using only $O(mk^2)$ space, which is a great advantage over the heuristic algorithms used for designing blue-noise masks in digital halftoning. This paper serves as a starting point of this type of investigation. A number of issues are still left open. One of the most interesting and attractive problems is to find low-discrepancy matrices, when n (dimension of the matrix) and k (the dimension of submatrix) are relatively prime to each other.

References

1. A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber: "Geometric Applications of a Matrix Searching Algorithm," Proc. 2nd ACM Symposium on Computational Geometry, pp. 285–292, 1986.
2. T. Asano, N. Katoh, K. Obokata, and T. Tokuyama, "Combinatorial and Geometric Problems Related to Digital Halftoning," *Theoretical Foundations of Computer Vision: Geometry, Morphology, and Computational Imaging*, LNCS 2616, Springer, 2003.
3. T. Asano, "Digital Halftoning: Algorithm Engineering Challenges," *IEICE Trans. on Inf. and Syst.*, E86-D, 2, 159–178, 2003.
4. T. Asano, K. Obokata, N. Katoh, and T. Tokuyama: "Matrix rounding under the L_p -discrepancy measure and its application to digital halftoning," *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 896–904, San Francisco, 2002.
5. B.E. Bayer: "An optimum method for two-level rendition of continuous-tone pictures," *Conference Record, IEEE International Conference on Communications*, 1, pp. (26-11)–(26-15), 1973.
6. B. Chazelle: *The Discrepancy Method: Randomness and Complexity*, Cambridge University Press, 2000.
7. R.W. Floyd and L. Steinberg: "An adaptive algorithm for spatial gray scale," *SID 75 Digest, Society for Information Display*, pp. 36–37, 1975.
8. H.D. Heinz: "Magic Squares, Magic Stars & Other Patterns" web site, <http://www.geocities.com/CapeCanaveral/Launchpad/4057/>.
9. R.L. Graham, B.D. Lubachevsky, K.J. Nurmela, P.R.J. Östergård: "Dense packings of congruent circles in a circle," *Discrete Math.*, 181, 139–154, 1998.
10. D.E. Knuth, *The Art of Computer Programming*, volume 1, *Fundamental Algorithms*, 3rd ed. (Reading, Massachusetts: Addison-Wesley, 1997).
11. Kodera Ed.: *Practical Design and Evaluation of Halftoned Images*, (in Japanese) Trikepps, 2000.
12. J. Matoušek: *Geometric Discrepancy*, Springer, 1991.
13. T. Mitsa and K.J. Parker: "Digital halftoning technique using a blue-noise mask," *J. Opt. Soc. Am.*, A/Vol. 9, No. 11, 1920–1929, 1992.
14. K.J. Nurmela and P.R.J. Östergård: "Packing up to 50 Equal Circles in a Square," *Discrete Comput. Geom.*, 18, 111–120, 1997.
15. R.A. Ulichney: "Dithering with blue noise," *Proc. IEEE*, 76, 1, 56–79, 1988.
16. R. Ulichney: "The void-and-cluster method for dither array generation," *IS&T/SPIE Symposium on Electronic Imaging Science and Technology, Proceedings of Conf. Human Vision, Visual Processing and Digital Display IV*, (Eds. Allebach, John Wiley), SPIE vol.1913, pp. 332–343, 1993.
17. M. Yao and K.J. Parker: "Modified approach to the construction of a blue noise mask," *J. Electronic Imaging*, 3, 92–97, 1994.

Voronoi Diagrams with a Transportation Network on the Euclidean Plane^{*}

Sang Won Bae and Kyung-Yong Chwa

Division of Computer Science, Department of EECS,
Korea Advanced Institute of Science and Technology, Daejeon, Korea
{swbae, kychwa}@jupiter.kaist.ac.kr

Abstract. This paper investigates geometric and algorithmic properties of the Voronoi diagram with a transportation network on the Euclidean plane. With a transportation network, the distance is measured as the length of the shortest (time) path. In doing so, we introduce a needle, a generalized Voronoi site. We present an $O(nm^2 \log n + m^3 \log m)$ algorithm to compute the Voronoi diagram with a transportation network on the Euclidean plane, where n is the number of given sites and m is the complexity of the given transportation network.

1 Introduction

With a transportation network, like streets in a city, subway or bus networks, or a highway over a nation, people have become capable to move faster. Finding a shortest (time) path using such transportation is very important due to the tendency of people trying to move with the shortest duration. In this situation, let us imagine a set of sites or service stations, and suppose that one should visit any of them; here, the Voronoi diagram problem arises. This paper considers geometric and algorithmic issues about the Voronoi diagram under the above situation.

The Voronoi diagram on the plane with a transportation network has been studied by several researchers. Hurtado et al. [1] and Abellanas et al. [2] discussed a single infinite road network as a straight line on the Euclidean plane; the currently best known results about the Voronoi diagram on the Euclidean plane with a transportation network. In addition, transportation networks on the L_1 plane have been dealt with; Abellanas et al. [3] dealt with isothetic and monotone networks and, more generally, Aichholzer et al. [4] introduced the Voronoi diagram with an isothetic transportation network on the L_1 plane.

All the previous work has assumed that every roads in a transportation network have the same speed. But, as mentioned at the beginning, several kinds of transportation networks are established on our surroundings and, hence, transportation networks with diverse speeds need to be considered. This paper is the first result dealing with transportation networks with multiple roads, possibly having arbitrary directions and diverse speeds, on the Euclidean plane.

^{*} This work is supported by grant No.R01-2003-000-11676-0 from KOSEF.

1.1 The Model

A transportation network consists of several roads as non-intersecting line segments on the plane, which may share a road, or an endpoint, in common. The roads have individual supporting speeds. This structure can be generally represented as a planar straight-line graph $G = (V, E)$ with speed v on each road, where V is a set of nodes and E is a set of roads.

In our model, one can enter into and exit from G at any point of roads or nodes. Along any road $e \in E$, one can move at fixed speed $v(e)$. Off the network G , one can move at unit speed in any direction. We assume that $v(e) > 1$ for all roads $e \in E$, since, if $v(e) \leq 1$, the road e does not contribute any shortest paths. Restricting entry points to fixed points on the roads yields an instance of Voronoi diagrams for additively weighted points, easily solvable (mentioned in [4]).

Given a transportation network G , the transportation metric d_G is measured as the length (duration) of the shortest (time) path when using roads in G . In fact, d_G induces a metric on \mathbf{R}^2 as a shortest path metric. We call the metric induced on \mathbf{R}^2 by d_G the transportation metric on G .

1.2 The Results

We present an algorithm for constructing the Voronoi diagram under the transportation metric with G , $\mathcal{V}_G(S)$, for a set S of n sites. The algorithm runs in $O(nm^2 \log n + m^3 \log m)$ time and requires $O(m(n + m))$ space, where m is the number of roads in G . Throughout this paper, we will denote the number of roads by m and the number of sites by n .

As a special case, we can reduce the runtime when the given transportation network is with the constraint that all roads have the same speed and their possible directions are finitely fixed. In this situation, we can compute the diagram in $O(nm \log n + m^2 \log m)$ time with linear space.

In doing so, we introduce a generalized site, a needle. A needle can be represented as a set of additively weighted points, lying on a line and weighted linearly. A Voronoi diagram for needles with a certain condition, so called *needle condition*, on the Euclidean plane can be computed in $O(n \log n)$ time and in $O(n)$ space. Arguments and details about needles are presented in Section 3.

Under the transportation metric, it is not so easy any more to plan a shortest path, even if a source and a destination are determined. We simply state our results about the path planning under the transportation metric in Section 6.

2 Basic Idea

In this section, we describe our basic idea to solve the problem through an observation on the plane with a transportation metric.

The transportation metric has some bad properties so that computing Voronoi diagrams under it is not so easy. For example, we can easily construct a cyclic

bisector with $\Omega(m)$ segments between two points; a transportation network with cyclic roads and two sites, one of which is much closer to a road than the other. This violates one of necessary conditions to apply the abstract Voronoi diagram, one of the most popular and useful tools to compute Voronoi diagrams (see Subsection 3.2). Thus, we should find a different approach to solve the problem.

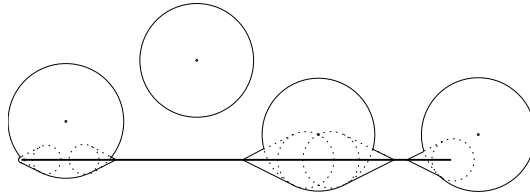


Fig. 1. Illustrations of t -neighborhood sets when G has a single road

To find such a different approach, we have to investigate the plane with a transportation network more carefully. Let us consider a simplest case, a single road transportation network. Given a single road on the Euclidean plane, we know how to find a shortest path by Hurtado et al. [1] and Abellanas et al. [2]. We draw the following observation from their results.

Observation 1.

Let e be a road segment of length l and let p be a point on the plane. Let α be the angle between the line segment pe and the road segment e . Then, the t -neighborhood set of p is the union of a disk of radius t centered at p and two needle-shaped regions along the road segment e of length $2t \sin \alpha$. Here, $\sin \alpha = 1/v(e)$.

By Observation 1, we can find the t -neighborhood set of p under the transportation metric with G , defined as $\{x | d_G(x, p) < t\}$, for any point p on the plane and $t \geq 0$. We shall denote it by $N_G(p, t)$. Figure 1 shows four illustrations for the boundary of $N_G(p, t)$ when G is a single road network.

Note that $N_G(p, t)$ can be represented as a union of one large disk centered at p and two needle-shaped regions along the road, each of which is a union of a set of disks whose radii are linearly assigned along the road. This provides an intuition to analyze the transportation metric from a different view. Our strategy is to consider such a needle-shaped region produced from p as an independent Voronoi site under the Euclidean metric. In Section 3, we define such a Voronoi site, called a *needle*. Then, we can compute the distance with more roads by using needles on the roads, if we can compute the distance to a needle from any point under the Euclidean metric. We will show how to compute a set of needles from given sites S , which allows us to properly analyze the given transportation network G , in Section 4 and 5.

3 Needle: A Generalized Site

In this section, we introduce a *needle* as a more generalized Voronoi site. Needles play a very important role in computing diagrams under the transportation metric. In Subsection 3.1, we define a needle formally and show that the distance from any point to a needle can be computed. In Subsection 3.2, we discuss Voronoi diagrams for needles on the Euclidean plane.

3.1 Definition of a Needle

A needle is generalized from a line segment with an additive weight. Note that a line segment can be viewed as a set P of points on it and the distance to a line segment from any point q on the plane is represented by $\min_{p \in P} d(p, q)$. Similarly, a needle is a set of weighted points lying on a line segment. Note that a weighted point p with weight w can be represented as a pair (p, w) and the distance to (p, w) from any point q is equal to $d(p, q) + w$. One necessary condition for a needle is that the weight on a needle is assigned linearly. With this property, a needle is suitable to represent $N_G(p, t)$, as noted in Section 2. The following is a definition of a needle.

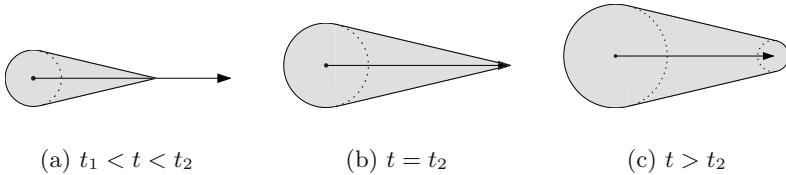


Fig. 2. t -neighborhood sets of a needle

Definition 1. A needle \mathbf{p} is a directed line segment $s(\mathbf{p})$ with endpoints $p_1(\mathbf{p})$ and $p_2(\mathbf{p})$ and a weight function $w_{\mathbf{p}}(x)$ defined on $s(\mathbf{p})$ such that $w_{\mathbf{p}}(p_1(\mathbf{p})) = t_1(\mathbf{p})$ and $w_{\mathbf{p}}(p_2(\mathbf{p})) = t_2(\mathbf{p})$. The t -neighborhood set of \mathbf{p} is denoted by $N(\mathbf{p}, t)$ for $0 \leq t_1 \leq t_2$.

Figure 2 shows t -neighborhood sets $N(\mathbf{p}, t)$ of a needle \mathbf{p} under the Euclidean metric, when $t_1 < t < t_2$, $t = t_2$ and $t > t_2$, case by case. As t increases from 0, $N(\mathbf{p}, t)$ grows from a point $p_1(\mathbf{p})$ at $t = t_1(\mathbf{p})$ and reaches $p_2(\mathbf{p})$ at $t = t_2(\mathbf{p})$ along $s(\mathbf{p})$. We also define the *speed* of a needle \mathbf{p} as $v(\mathbf{p}) = d(p_1(\mathbf{p}), p_2(\mathbf{p})) / (t_2(\mathbf{p}) - t_1(\mathbf{p}))$ and its *direction* as direction toward $p_2(\mathbf{p})$ from $p_1(\mathbf{p})$. Note that if $v(\mathbf{p}) \leq 1$, $N(\mathbf{p}, t)$ is of shape of a disk, and thus we can regard \mathbf{p} as a point $p_1(\mathbf{p})$ with weight $t_1(\mathbf{p})$.

Throughout this paper, we will draw a needle as an arrow headed in its direction, as shown in Figure 2.

We now discuss the distance from any point on the plane to a needle under the Euclidean metric. For any point $x \in \mathbf{R}^2$ and any needle \mathbf{p} , the distance is represented as follows:

$$d(x, \mathbf{p}) = \min_{y \in s(\mathbf{p})} \{d(x, y) + w_{\mathbf{p}}(y)\},$$

where $w_{\mathbf{p}}(y)$ is the weight assigned to y . $w_{\mathbf{p}}(y)$ can be computed by the definition of a needle as follows:

$$\begin{aligned} w_{\mathbf{p}}(y) &= t_1(\mathbf{p}) + (t_2(\mathbf{p}) - t_1(\mathbf{p})) \frac{d(p_1(\mathbf{p}), y)}{d(p_1(\mathbf{p}), p_2(\mathbf{p}))} \\ &= t_1(\mathbf{p}) + d(p_1(\mathbf{p}), y)/v(\mathbf{p}), \end{aligned}$$

for all $y \in s(\mathbf{p})$.

We define some terms associated with a needle \mathbf{p} . Without loss of generality, we assume that \mathbf{p} is horizontal and p_1 is to the left of p_2 . Let l_1 denote a line, whose slope is $\tan \alpha$ and which meets p_1 , and l_2 denote a line, whose slope is $\tan(\pi/2 + \alpha)$ and which meets p_2 , where $\alpha = \sin^{-1}(1/v(\mathbf{p}))$. We define $s^+(\mathbf{p})$ to be a line segment associated with \mathbf{p} whose endpoints are p_1 and the intersection point between l_1 and l_2 . Symmetrically, we define $s^-(\mathbf{p})$; see Figure 3. Next, we let \mathbf{p}^+ be a set of points above $s(\mathbf{p})$ such that, for any point $x \in \mathbf{p}^+$, a line, which is perpendicular to $s^-(\mathbf{p})$ and meets x , intersects $s^-(\mathbf{p})$. In this manner, we also define \mathbf{p}^- , symmetrically. Note that \mathbf{p}^+ and \mathbf{p}^- decompose \mathbf{R}^2 into four disjoint and connected regions as shown in Figure 3.

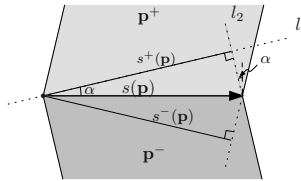


Fig. 3. Plane decomposition around a needle \mathbf{p}

Lemma 1. For any needle $\mathbf{p} = (p_1, p_2, t_1, t_2)$ and any point $x \in \mathbf{R}^2$,

$$d(x, \mathbf{p}) = \begin{cases} d(x, s^-(\mathbf{p})) + t_1 & x \in \mathbf{p}^+ \\ d(x, s^+(\mathbf{p})) + t_1 & x \in \mathbf{p}^- \\ \min\{d(x, p_1) + t_1, d(x, p_2) + t_2\} & \text{else} \end{cases}$$

By the above lemma, we can easily compute the distance from any point to a needle in constant time.

3.2 Voronoi Diagrams for Needles Under the Euclidean Metric

In this subsection, we compute the Voronoi diagram for a set of needles. In order to do so, we recall the model introduced by Klein [5]. In this model, a system of bisecting curves for S , $(S, \{J(p, q) | p, q \in S, p \neq q\})$ is given, which is called a *needle Voronoi diagram* if the following conditions are fulfilled:

1. $J(p, q)$ is homeomorphic to a line or empty.
2. The intersection of any two bisectors consists of finitely many components.
3. $R(p, q) \cap R(q, r) \subset R(p, r)$.
4. For any subset $S' \subseteq S$ and $p \in S'$, $R(p, S')$ is path-connected if it is nonempty.

where $R(p, q) = \{x \in \mathbf{R}^2 \mid d(x, p) < d(x, q)\}$, $R(p, S) = \bigcap_{q \in S, p \neq q} R(p, q)$ and $J(p, q)$ is the bisecting region between p and q .

However, needles violate condition 1 and condition 4, since needles generalize line segments. Note that the bisector between two needles \mathbf{p} and \mathbf{q} is defined as $J(\mathbf{p}, \mathbf{q}) = \{x \mid d(x, \mathbf{p}) = d(x, \mathbf{q})\}$, which is computable in constant time by Lemma 1.

We suggest another condition restricting needles so that the bisector system is admissible. We consider the following four cases for each needle \mathbf{p} with respect to another needle \mathbf{q} .

1. $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q}) = \emptyset$
2. $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q}) \neq \emptyset$ and $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q})$ includes either $p_1(\mathbf{q})$ or $p_2(\mathbf{q})$, not both.
3. $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q}) \neq \emptyset$ and $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q})$ includes both $p_1(\mathbf{q})$ and $p_2(\mathbf{q})$.
4. $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q}) \neq \emptyset$ and $R(\mathbf{p}, \mathbf{q}) \cap s(\mathbf{q})$ includes neither $p_1(\mathbf{q})$ nor $p_2(\mathbf{q})$.

We call two needles, \mathbf{p} and \mathbf{q} , *non-piercing* if and only if, for each with respect to the other, the fourth case does not occur. Otherwise, we call two needles *piercing*. Now, we consider only a set of pairwise non-piercing needles that leads to our goal, sufficiently. Note that the non-piercing for needles condition generalizes the disjointness for line segments.

Theorem 1. *Let S be a set of pairwise non-piercing needles. Then, the bisector system $(S, \{J(\mathbf{p}, \mathbf{q}) \mid \mathbf{p}, \mathbf{q} \in S, \mathbf{p} \neq \mathbf{q}\})$ is admissible.*

Constructing abstract Voronoi diagrams has been dealt with by several researchers, Klein [5], Mehlhorn et al. [6], Klein et al. [7], and Dehne and Klein [8]. All presented algorithms take an optimal time and an optimal space. Thus, we conclude the following corollary as a result of this section.

Corollary 1. *Let S be a set of pairwise non-piercing needles. Then, the bisector system $(S, \{J(\mathbf{p}, \mathbf{q}) \mid \mathbf{p}, \mathbf{q} \in S, \mathbf{p} \neq \mathbf{q}\})$ is admissible and the complexity is $O(n \log n)$.*

In fact, Aichholzer et al.[4] introduced the concept similar to non-piercing needles. The authors discussed their concepts in the wavefront model and they defined a needle as a figure produced by the wavefront of a site while touching road segments. In the wavefront model, each site p sends out a wavefront and the interference pattern of these wavefronts contributes the diagram. In contrast with their approach, we define a needle as an independent Voronoi site.

4 Relation Between a Transportation Network and Needles

In this section, we draw a relation between a given transportation network and a set of needles.

Theorem 2.

$$R(\mathbf{p}, \mathcal{S}) \subseteq R_G(\mathbf{p}, S) \quad \text{for } \mathbf{p} \in \mathcal{S}, \quad \text{and} \quad R(\mathbf{p}, \mathcal{S}) \subseteq R_G(q, S) \quad \text{for } q \in S.$$

First, a point p produces at most two new needles on a road e (see Figure 1). These needles are born at a node of e or at an intersection point of the road with the line, whose slope is $\tan(\pi/2 \pm \alpha)$ and which meets p by Observation 1. One important fact is that the produced needles are also Voronoi sites and thus they possibly produce new needles on other roads. We will call such a newly produced needle \mathbf{q} .

Note that, generally, a needle produces at most two new needles on a road. Suppose that a needle produces three needles on a road. Then, two of the three should be in the same direction and one of these two needles must be dominated by the original needle or other newly produced ones, since the speeds of the three needles are the same and their neighborhood sets are convex under the Euclidean metric (see Figure 1 and 2).

More formally, we define $\sigma_G(\mathbf{p})$ to be a set of needles produced from a needle \mathbf{p} on all roads in G . For a set X of needles, $\sigma_G(X) = \bigcup_{\mathbf{p} \in X} \sigma_G(\mathbf{p})$. In order to obtain all needles produced from a site p , we apply $\sigma_G(\cdot)$, repeatedly. We let $\sigma_G^k(p) = \sigma_G(\sigma_G^{k-1}(p))$ and $\sigma_G^0(p) = \{p\}$. We then let \mathcal{S}_p^k be $\bigcup_{i=0}^k \sigma_G^i(p)$ and \mathcal{S}_p be \mathcal{S}_p^∞ . Moreover, we let $d_G^k(p, q)$ be the length of a shortest path from p to q where the path passes through at most k roads. Surely, $d_G(p, q) = d_G^\infty(p, q)$.

We claim that $d_G^k(p, q) = d(p, \mathcal{S}_q^k)$, which directly implies the theorem.

There exists a needle $\mathbf{q} \in \mathcal{S}_q^k$ such that $d(p, \mathbf{q}) = d(p, \mathcal{S}_q^k)$ and $\mathbf{q} \in \sigma_G^l(q)$, for any point p and $0 \leq l \leq k$. Let us consider a sequence of needles, $\mathbf{q}_l = \mathbf{q}, \mathbf{q}_{l-1}, \dots, \mathbf{q}_1$, such that \mathbf{q}_i is produced from \mathbf{q}_{i-1} and $\mathbf{q}_i \in \sigma_G^i(q)$. With these needles, we construct a path \mathcal{P} from p to q passing through l roads. \mathcal{P} starts p and goes to the points y_l on $s(\mathbf{q}_l)$ which obeys the rule of Observation 1. Next, \mathcal{P} passes $p_1(\mathbf{q}_l)$ along the road, on which \mathbf{q}_l is produced. Recursively, \mathcal{P} passes y_i and $p_1(\mathbf{q}_i)$ next to $p_1(\mathbf{q}_{i+1})$ and finally reaches q from $p_1(\mathbf{q}_1)$. Since each \mathbf{q}_i is produced on a road, \mathcal{P} passes through l roads, which is at most k , hence, $d_G^k(p, q) \leq d(p, \mathcal{S}_q^k)$.

Let us consider a path \mathcal{P} of length $d_G^k(p, q)$ from p to q and assume that \mathcal{P} passes through l roads, for $0 \leq l \leq k$. We then construct l needles, $\mathbf{q}_1, \dots, \mathbf{q}_l$, inductively, such that $\mathbf{q}_i \in \sigma_G^i(q)$ and $d(p, \mathbf{q}_l) = d_G^k(p, q)$.

\mathcal{P} consists of l segments on roads, e_l, e_{l-1}, \dots, e_1 , in order, each of which starts at y_i and ends at x_i on road e_i . Also, v_i denotes a node of e_i in direction toward y_i from x_i . Note that roads indicated by e_i may not be disjoint. We construct \mathbf{q}_i as follows: $\mathbf{q}_1 = (x_1, v_1, d(x_1, q), d(x_1, q) + \frac{1}{v(e_1)}d(x_1, v_1))$ and $\mathbf{q}_i = (x_i, v_i, d(x_i, \mathbf{q}_{i-1}), d(x_i, \mathbf{q}_{i-1}) + \frac{1}{v(e_i)}d(x_i, v_i))$. For convenience, let \mathbf{q}_0 be

q . Then, $\mathbf{q}_i \in \sigma_G^i(q)$, since \mathcal{P} is a shortest path and it obviously obeys the rule of Observation 1. We are sure that $d(x_1, \mathbf{q}_0) = d_G^0(x_1, q) = d(x_1, q)$. If $d(x_i, \mathbf{q}_{i-1}) = d_G^i(x_i, q)$, a sub-path \mathcal{P}' of \mathcal{P} from x_i to q is a shortest path passing through i roads. We construct a path \mathcal{P}'' of length $d(x_{i+1}, \mathbf{q}_i)$ from x_{i+1} to q . \mathcal{P}'' is obtained by adding the path $x_{i+1} \rightarrow y_i \rightarrow x_i$ at the beginning of \mathcal{P}' . Then, $d(x_{i+1}, \mathbf{q}_i) = d_G^{i+1}(x_{i+1}, q)$, since \mathcal{P}'' passes through $i + 1$ roads and a sub-path of \mathcal{P} . By induction, $d(p, \mathbf{q}_l) = d_G^l(p, q) = d_G^k(p, q)$, implying that $d_G^k(p, q) \geq d(p, S_q^k)$.

Finally, we show that $d_G^k(p, q) = d(p, S_q^k)$, and thus $d_G(p, q) = d(p, S_q)$. With taking $\mathcal{S} = \bigcup_{p \in S} S_p$, this says that, for any $\mathbf{p} \in \mathcal{S}$, there exists a site $q \in S$ such that $R(\mathbf{p}, \mathcal{S}) \subseteq R_G(q, S)$. □

Theorem 2 shows the existence of a set of needles noted at the end of Section 2. But, \mathcal{S} may not be pairwise non-piercing and further have infinitely many needles by the construction from the proof of Theorem 2. Thus, we should reduce the number of needles and make them pairwise non-piercing. For any $\mathbf{p} \in \mathcal{S}$, we call \mathbf{p} *effective* if $R(\mathbf{p}, \mathcal{S}) \neq \emptyset$, otherwise we call \mathbf{p} *ineffective*.

Lemma 2. \mathcal{S}^* $\mathcal{V}(\mathcal{S}^*) = \mathcal{V}(\mathcal{S})$

First, we remove all ineffective needles from \mathcal{S} . We let $\mathcal{S}_e \subseteq \mathcal{S}$ be the set of all effective needles in \mathcal{S} . Trivially, $\mathcal{V}(\mathcal{S}_e) = \mathcal{V}(\mathcal{S})$. We will construct \mathcal{S}^* from \mathcal{S}_e by making it pairwise non-piercing without any change in its diagram.

We can partition \mathcal{S} into $\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_m$ such that $\mathcal{S}_0 = \{(p, p, 0) | p \in S\}$ and all needles in \mathcal{S}_i lie on e_i , for $1 \leq i \leq m$, if all roads in G are given as $\{e_1, \dots, e_m\}$. For each needle $\mathbf{p} \in \mathcal{S}_i$, $1 \leq i \leq m$, we can find (at most) one needle, $\mathbf{q} \in \mathcal{S}_i$, the closest needle in the direction of \mathbf{p} from $p_1(\mathbf{p})$. \mathbf{p} stretches to a node of e_i but the part over $p_1(\mathbf{q})$ does not contribute $\mathcal{V}(\mathcal{S})$, since both \mathbf{p} and \mathbf{q} are effective and their speeds are the same as $v(e_i)$. We thus cut \mathbf{p} by setting $p_2(\mathbf{p})$ to be $p_1(\mathbf{q})$ and $t_2(\mathbf{p})$ to be $t_1(\mathbf{p}) + \frac{1}{v(e_i)}d(p_1(\mathbf{p}), p_1(\mathbf{q}))$. We denote the set of these new needles from \mathcal{S}_i by \mathcal{S}_i^* and $\mathcal{S}_0 \cup \mathcal{S}_1^* \cup \dots \cup \mathcal{S}_m^*$ by \mathcal{S}^* . Since we cut only a useless part of every needle from \mathcal{S}_e , $\mathcal{V}(\mathcal{S}^*) = \mathcal{V}(\mathcal{S}_e)$.

Now, we show that \mathcal{S}^* is pairwise non-piercing. \mathcal{S}_0 is surely pairwise non-piercing, since it is a set of points. \mathcal{S}_i^* is also pairwise non-piercing, since all needles in \mathcal{S}_i^* lie on a line including a road e_i and they are disjoint.

We assume that \mathbf{p} pierces \mathbf{q} in \mathcal{S}^* and \mathbf{q} is produced on a road e_i . \mathbf{q} is not a needle produced from \mathbf{p} , since \mathbf{p} and \mathbf{q} are non-piercing if \mathbf{q} is a needle produced from \mathbf{p} . If a needle \mathbf{p}' produced on e_i from \mathbf{p} is effective, \mathbf{q} cannot dominate any point on e_i over $p_1(\mathbf{p}')$, and thus \mathbf{q} is not pierced. Hence, no needles on e_i from \mathbf{p} are effective. This implies that \mathbf{q} dominates two possible starting points of needles produced from \mathbf{p} , and thus \mathbf{q} dominates all points between the two points. Consequently, \mathbf{p} does not pierce \mathbf{q} , a contradiction. Therefore, \mathcal{S}^* is pairwise non-piercing. □

5 Algorithm

In this section, we consider more algorithmic issues. A simple algorithm for computing \mathcal{S}^* from \mathcal{S} is presented. The algorithm runs in $O(nm^2 \log n + m^3 \log m)$ time.

5.1 Computing \mathcal{S}^*

We recall the wavefront model. In the wavefront model, each site starts sending out a wavefront at time 0 and the interference pattern between each wavefront constitutes the Voronoi diagram. A wavefront at time t from each site p can be interpreted as the set $\{x | d_G(x, p) = t\}$, the boundary of $N_G(p, t)$ under the transportation metric with G .

It is too hard to compute \mathcal{S}^* from \mathcal{S} , since \mathcal{S} can include possibly infinitely many needles, as noted earlier. However, the effectiveness of a needle \mathbf{p} can be tested only with effective needles already emitting wavefronts at time $t_1(\mathbf{p})$. Thus, we simulate the wavefront model with sweeping the plane with the wavefronts. During simulating the wavefront model, at time $t_1(\mathbf{p})$, we test the effectiveness of \mathbf{p} and we do some processes for \mathbf{p} , only if \mathbf{p} is effective. In this way, we compute \mathcal{S}^* directly, not from \mathcal{S} . Lemma 4 reduces the number of applying $\sigma_G(\cdot)$.

Lemma 3.

Lemma 4.

Our algorithm runs with handling \dots , defined as certain situations in the wavefront model. Two kinds of events are defined; one occurs when a needle produced on a road from another effective needle starts growing, which we call a \dots ; and the other occurs when a wavefront sent out from a needle reaches a node, called a \dots . From each event, we can ascertain the occurring time, the occurring place, the associated site, and so on. We will call an event \dots if its associated needle dominates the point where the event occurs. To handle these events, we need two types of data structures:

- \mathcal{Q} is an event queue implemented as a priority queue. The priority of an event e is its occurring time. \mathcal{Q} supports inserting, deleting, and extracting-minimum in logarithmic time with linear space.
- $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_m$ are balanced binary search trees, each associated with e_i , where the road set E is given as $\{e_1, e_2, \dots, e_m\}$. Each \mathcal{T}_i stores needles on e_i in order. The precedence is the position of the starting point of a needle and ties are broken by the growing direction. \mathcal{T}_i supports inserting and deleting of a needle in logarithmic time, and also a linear scan for needles currently in \mathcal{T} in linear time with linear space.

At the beginning of the algorithm, we initialize \mathcal{Q} and \mathcal{T}_i , for $1 \leq i \leq m$. We then compute the associating birth events with $\sigma_G(p)$ for all $p \in S$ and insert them into \mathcal{Q} . And, for each node $v \in V$, we compute the node event for the nearest site from v , insert it into \mathcal{Q} , and set the variable $event(v)$ to be the node event.

Once the initialization is done, we extract the upcoming event from \mathcal{Q} and process it as follows, repeatedly, while \mathcal{Q} is not empty.

- If a birth event b on e_i occurs, we first test the effectiveness of its associated needle \mathbf{p} , and then insert it to \mathcal{T}_i . The effectiveness checking can be accomplished by checking with at most two neighbor needles in \mathcal{T}_i . More precisely, we find two neighbors around $p_1(\mathbf{p})$ from \mathcal{T}_i , and test whether two neighbors dominate $p_1(\mathbf{p})$. (Note that this test is necessary but not sufficient, i.e. ineffective needles may pass this test. This however does not increase the asymptotic number of needles; we will discuss about this argument in the next subsection.) Once the effectiveness test is passed, we insert \mathbf{p} into \mathcal{T}_i and compute node events from \mathbf{p} . Note that $event(v)$ is the currently earliest node event on v in \mathcal{Q} . We let a be the node event on v newly computed from \mathbf{p} . Then, only if a will occur earlier than $event(v)$, we delete $event(v)$ from \mathcal{Q} , insert a into \mathcal{Q} and set $event(v)$ as a .
- When a node event a on a node v occurs, we compute $\sigma_G(\mathbf{p})$, where \mathbf{p} is the needle associated with a . Then, from all needles in $\sigma_G(\mathbf{p})$, we compute the associating birth events and insert them into \mathcal{Q} .

After the event processing, all effective needles are stored in each \mathcal{T}_i . We then extract them with proper cutting, as described in the proof of Lemma 2.

We denote by \mathcal{S}_a^* the resulting set of needles after running the algorithm and by \mathcal{S}_a $S \cup \bigcup_{1 \leq i \leq m} \mathcal{T}_i$. The following lemma guarantees the correctness of the algorithm.

Lemma 5. $\mathcal{S}_e \subseteq \mathcal{S}_a \subseteq \mathcal{S} \subseteq \mathcal{S}_a^* \subseteq \mathcal{S}$
 $\mathcal{S}_e \subseteq \mathcal{S}_a \subseteq \mathcal{S}.$

During running the algorithm, all needles inserted into \mathcal{T}_i are obtained by applying $\sigma_G(\cdot)$ from any site $p \in S$, which implies that $\mathcal{S}_a \subseteq \mathcal{S}$.

The algorithm filters ineffective needles efficiently by Lemma 4 and by the effectiveness testing in processing a birth event, which is necessary. Hence, all needles in $\mathcal{S} \setminus \mathcal{S}_a$ are ineffective and $\mathcal{S}_e \subseteq \mathcal{S}_a$.

Note that all ineffective needles are totally dominated by other effective ones. To effective needles in \mathcal{S}_a^* , we can apply an analogue to the proof of Lemma 4. Thus, \mathcal{S}_a^* is pairwise non-piercing. □

5.2 Analysis

Our algorithm depends on the number of handled events and on the number of needles in \mathcal{S}_a^* . In fact, the number of events is $O(m(n + m))$ and also $|\mathcal{S}_a^*| =$

$O(m(n+m))$, since the algorithm applies $\sigma_G(\cdot)$ exactly $n+m$ times and $|\sigma_G(\mathbf{p})| = O(m)$ for any needle \mathbf{p} . This implies the following lemma.

Lemma 6.

$$|\mathcal{S}_a^*| = O(m(n+m)) \quad \dots \quad |\mathcal{S}^*| = O(m(n+m)),$$

\dots n \dots m \dots G

Finally, we conclude a main theorem.

Theorem 3. \dots G \dots m \dots S \dots
 n \dots $\mathcal{V}_G(S)$ \dots S \dots
 \dots G \dots $O(nm^2 \log n + m^3 \log m)$ \dots $O(m(n+m))$ \dots

Handling an event takes $O(m \log(n+m))$ time. Thus, computing \mathcal{S}_a^* takes $O(m^2(n+m) \log(n+m))$ time and this is a bottleneck of the algorithm. For any other processes of the algorithm, $O(m(n+m) \log(n+m))$ time is sufficient.

For the space complexity, the algorithm uses only a linear space on the number of handled events and produced needles, which is $O(m(n+m))$. \square

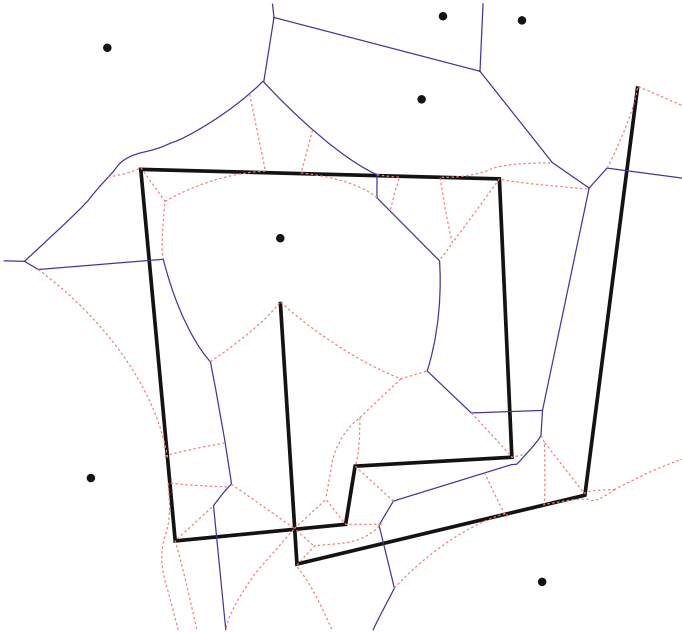


Fig. 4. Illustration of a Voronoi diagram $V_G(S)$ with the transportation network on the Euclidean plane. Dashed lines depict Voronoi edges in $V(S)$

6 Concluding Remarks

In previous sections, we consider only a set of point sites but our framework is also applicable to different types of sites, since we handle needles, more generalized sites. We showed that, for points additive weight or for disjoint line segments, our algorithm does work properly with minor modification and no gains for the runtime.

As mentioned in the introduction, it is not trivial to plan a shortest path under the transportation metric. We also showed that a shortest path to the nearest site from a given point can be reported in $O(\log(n + m) + r)$ time by point location over $\mathcal{V}(\mathcal{S}^*)$, where r is the path complexity, bounded by $O(m)$. And a shortest path between given two points can be reported in $O(m^3 \log m)$ time by constructing the shortest path map, in fact, the Voronoi diagram of needles produced from one of the two given points; this is a special case of .

by Mitchell and Papadimitriou [9], which is solved in $O(N^8 L)$ time where N is the complexity of the polygonal partition and L denotes the precision of the problem instance.

Our algorithm is easy to implement (We have implemented it in practice; Figure 4 is obtained by our program) but the asymptotical complexity is not optimal. A future work is to improve the algorithm in runtime while retaining the ease of implementation.

References

1. Hurtado, F., Palop, B., Sacristán, V.: Diagramas de voronoi con funciones temporales. VIII Encuentros en Geometria Computacional (1999)
2. Abellanas, M., Hurtado, F., Sacristán, V., Palop, B.: Voronoi diagram for services neighboring a highway. IPL: Information Processing Letters **86** (2003)
3. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: Proximity problems for time metrics induced by the l_1 metric and isothetic networks. IX Encuentros en Geometria Computacional (2001)
4. Aichholzer, O., Aurenhammer, F., Palop, B.: Quickest paths, straight skeletons, and the city voronoi diagram. In: Proceedings of the 8th SoCG, (June, 5-7, 2002), New York, ACM Press (2002) 151–159
5. Klein, R.: Concrete and Abstract Voronoi Diagrams. Number 400 in Lecture Notes in Computer Science, LNCS. Springer-Verlag, Berlin, Germany (1989)
6. Mehlhorn, K., Meiser, S., O'Dunlaing, C.: On the construction of abstract Voronoi diagrams. Discrete Comput. Geom. **6** (1991) 211–224
7. Klein, R., Mehlhorn, K., Meiser, S.: Randomized incremental construction of abstract voronoi diagrams. Computational Geometry: Theory and Applications **3** (1993) 157–184
8. Dehne, F., Klein, R.: “the big sweep”: On the power of the wavefront approach to Voronoi diagrams. Algorithmica **17** (1997) 19–32
9. Mitchell, J.S.B., Papadimitriou, C.H.: The weighted region problem: Finding shortest paths through a weighted planar subdivision. Journal of the ACM **38** (1991) 18–73

Structural Alignment of Two RNA Sequences with Lagrangian Relaxation

Markus Bauer and Gunnar W. Klau

Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Austria
{markus, gunnar}@ads.tuwien.ac.at

Abstract. RNA is generally a single-stranded molecule where the bases form hydrogen bonds within the same molecule leading to structure formation. In comparing different homologous RNA molecules it is usually not sufficient to consider only the primary sequence, but it is important to consider both the sequence and the structure of the molecules. Traditional alignment algorithms can only account for the sequence of bases, but not for the base pairings. Considering the structure leads to significant computational problems because of the dependencies introduced by the base pairings and the presence of pseudoknots. In this paper we address the problem of optimally aligning two given RNA sequences either with or without known structure (allowing for pseudoknots). We phrase the problem as an integer linear program and then solve it using Lagrangian relaxation. In our computational experiments we could align large problem instances—18S and 23S ribosomal RNA with up to 1500 bases within minutes while preserving pseudoknots.

1 Introduction

Unlike DNA, an RNA molecule is generally single-stranded and folds in space due to the formation of hydrogen bonds between its bases. While similarity between two nucleic acid chains is usually determined by sequence alignment algorithms, these can only account for the primary structure and thus ignore structural aspects. However, the problem of producing RNA alignments that are structurally correct has emerged as one of the central obstacles for the computational study of functional RNAs. To date, the available tools for computing structural alignments are either based on heuristical approaches and thus produce suboptimal alignments or cannot attack instances of reasonable input size.

In this paper we deal with the comparison of two RNA sequences together with their structure. Indeed, we do not necessarily require the actual knowledge of either structure, but will infer a common structure based on the computation of preserved hydrogen bonds. In the presence of pseudoknots, the problem becomes NP-hard even when only two sequences have to be aligned (Evans gives an NP-hardness proof for a special case of this problem in [7]).

The computational problem of considering sequence and structure of an RNA molecule simultaneously was first addressed by Sankoff [15] who proposed a dynamic programming algorithm that aligns a set of RNA sequences while at

the same time predicting their common fold. Algorithms similar in spirit were proposed later on for the problem of comparing one RNA sequence to one or more of known structure. Corpet and Michot [5] align simultaneously a sequence with a number of other, already aligned, sequences using both primary and secondary structure. Their dynamic programming algorithm requires $\mathcal{O}(n^5)$ running time and $\mathcal{O}(n^4)$ space (n is the length of the sequences) and thus can handle only short sequences. They propose an anchor-point heuristic to divide large alignment problems by fixed alignment regions into small subproblems that the dynamic programming algorithm can then be applied to.

Bafna [2] improved the dynamic programming algorithm to a running time of $\mathcal{O}(n^4)$ which still does not make it applicable to real-life problems. Common motifs among several sequences are searched by Waterman [16]. Eddy and Durbin [6] describe probabilistic models for measuring the secondary structure and primary sequence consensus of RNA sequence families. They present algorithms for analyzing and comparing RNA sequences as well as database search techniques. Since the basic operation in their approach is an expensive dynamic programming algorithm, their algorithms cannot analyze sequences longer than 150-200 nucleotides. Gorodkin [8] and Mathews and Turner [12] published simplified versions of Sankoff's original algorithm.

Hofacker [10] give a different approach to structural alignments: instead of folding and aligning sequences simultaneously, they present a dynamic programming approach to align the corresponding secondary structures, computed by McCaskill's partition function algorithm [13], and therefore take the structural information into account.

Reinert [11] presented a Branch-and-Cut algorithm for aligning an RNA molecule of known sequence and known structure to an RNA molecule of known sequence but unknown structure. The algorithm computes an (optimal) alignment that maximizes sequence and structure consensus simultaneously and is based on the Branch-and-Cut technique. The method can handle pseudoknots and is able to solve already problems up to a size of 1400 bases. However, for problems of that size their implementation starts to require prohibitive time.

We start with a similar integer linear programming (ILP) formulation as given by Reinert, but instead of using a Branch-and-Cut approach, we resort to the ILP approach which was already successfully applied by Lancia to the contact map problem [4] which is similar to the RNA alignment problem.

The structural alignment of RNA molecules is central to the various RNA similarity or structure prediction problems defined in the above cited papers. If the two molecules are functionally related and have a similar structure, the RNA structural alignment allows, to draw conclusions about the structure of the unknown molecule. The techniques we will put forward can be modified such that incremental or simultaneous computations of multiple structural sequence alignments are possible.

Since the traditional approaches cannot solve middle sized or large instances of the structural RNA alignment problem without using heuristics, biologists still have to carry out large structural alignments by hand. Furthermore,

most algorithms are not able to integrate tertiary structure interactions like pseudoknots, or need prohibitive resources.

We tested a first version of our proposed algorithm with 23S rRNA sequences from [17] and 18S rRNA sequences from *Saccharomyces cerevisiae* (1995 bases) and human (1870 bases). We could, for example, optimally align the 23S rRNA sequences of *Saccharomyces cerevisiae* (1497 bases) and *Saccharomyces cerevisiae* (1495 bases). The result was a structural alignment that is a very good approximation of the “hand-made” optimal structural alignment and is, in particular, much better than the purely sequence-based optimal alignment. In the 18S rRNA dataset it is well known that the first 1200 bases contain pseudoknots. We can show that our approach correctly preserves these pseudoknots. To our knowledge, no other algorithm is capable of doing this.

The paper is structured as follows: Section 2 gives basic definitions and a mathematical formulation of the problem. In Sect. 3 we study how to relax the ILP in order to make it efficiently solvable, whereas Sect. 4 shows how we solve the original problem by means of Lagrangian relaxation. The results of our computational experiments are given in Sect. 6. Finally, we discuss our results in Sect. 7.

2 Basic Definitions and Mathematical Formulation

Before presenting a graph-theoretic model of aligning two RNA sequences and a corresponding (ILP) formulation we start with some basic definitions:

Definition 1. Let $S = (A, G, U, C, -)$ and $\Sigma = (s_1, \dots, s_n)$ be an RNA sequence and an alphabet, respectively. Let $(i, j) \in A$ be an interaction, $s_i \neq s_j$ be a conflict, and (S, P) be an annotated sequence.

Note that a structure where no pair of interactions is in conflict with each other forms a valid secondary structure of an RNA sequence.

We are given two annotated sequences (S_1, P_1) and (S_2, P_2) . In graph-theoretic terms the input can be modeled as a graph $G = (V, A \cup I)$ where the set V denotes the vertices of the graph, in this case the letters of the two sequences, a set A of edges between vertices of the two input sequences (the interactions) and I the set of interactions between vertices of the same sequence. The left side of Fig. 1 shows such an input graph. Dashed lines are interaction edges, solid lines are alignment edges.

Two alignment edges (a_1, b_1) and (a_2, b_2) are said to be in conflict, if $a_1 < a_2 \rightarrow b_1 < b_2$ or $a_1 > a_2 \rightarrow b_1 > b_2$ is not satisfied. Visually stated, alignment edges that are in conflict cross or touch each other. A subset \mathcal{A} of A is called an alignment, if no alignment edges are in conflict. Graph-theoretically, an alignment is a non-crossing matching.

Two interaction edges $i = (i_1, i_2) \in P_1$ and $j = (j_1, j_2) \in P_2$ are said to be realized by an alignment \mathcal{A} if and only if the alignment edges (i_1, j_1) and (i_2, j_2)

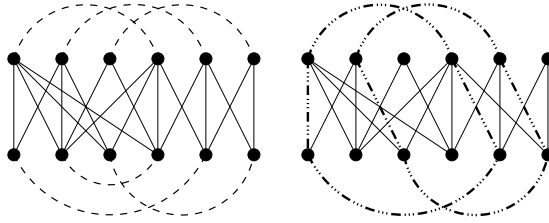


Fig. 1. Input graph for structural alignments and realized interaction matches

are realized by \mathcal{A} . The pair (i, j) is called an *interaction match*. Note that (i, j) is an ordered tuple, that is, (i, j) is distinct from (j, i) . The right side of Fig. 1 shows four interaction matches that are realized by the alignment (indeed it shows a preserved pseudoknot).

Each alignment edge and interaction match is assigned a positive weight representing the benefit of realizing this edge or the match. In the case of interaction edges we could choose the score for realizing the interaction match (i, j) , $i, j \in A$, as the number of hydrogen bonds between the bases or the base pair probability computed by means of McCaskill’s algorithm [13].

Traditional sequence alignments aim at maximizing the score of edges realized by an alignment. A structural alignment, however, takes the structural information (the information contained within the interaction edges) into account as well. The problem of structurally aligning two annotated sequences (S_1, P_1) and (S_2, P_2) calls for an optimal solution of $\max \sum_{a \in A} w_a + \sum_{i \in P_1, j \in P_2} w_{ij}$, that is, the score achieved by the weight of the alignment \mathcal{A} and realized interaction edges. Three properties, however, have to be satisfied in order to form a valid RNA secondary structure:

1. Every vertex is incident to at most one interaction edge.
2. Each interaction match has to be realized.
3. No alignment edges are in conflict.

Then, an ILP formulation follows directly:

$$\max \sum_{m \in A} \sum_{l \in A} w_{lm} y_{lm} + \sum_{m \in A} w_m x_m \tag{1}$$

$$\sum_{l \in I} x_l \leq 1 \tag{2} \quad \forall I \in \mathcal{I}$$

$$y_{lm} = y_{ml} \tag{3} \quad \forall l, m \in A, l < m$$

$$\sum_{l \in A} y_{lm} \leq x_m \tag{4} \quad \forall m \in A$$

$$x, y \geq 0 \tag{5} \quad \text{integer}$$

The variable x_m equals one, if alignment edge m is part of the alignment, whereas $y_{lm} = 1$ holds, if the alignment edges l and m realize the interaction

match (l, m) . The set \mathcal{I} contains all subsets of alignment edges, such that all pairs of elements of a specific subset are crossing each other. One can easily verify that all properties for a structural alignment are satisfied: inequalities (3) and (4) guarantee that interaction matches are realized by alignment edges and that every node is incident to at most one interaction edge, whereas (2) guarantees the alignment edges to be non-crossing.

The order $l < m$ within the equality constraints (3) denotes an arbitrary order defined on the elements of A (otherwise identical constraints show up twice, that is, $y_{ml} = y_{lm}$ and $y_{lm} = y_{ml}$ were part of the ILP). Due to the NP-hardness of the problem, we cannot hope to solve the ILP above directly. Therefore, we drop some constraints and show how the relaxed ILP can be solved efficiently.

3 Relaxation and Efficient Solution

We call the ILP (1)-(5) without the constraints (3) the *relaxed ILP* and show how to solve it efficiently. Later we show how to incorporate the dropped constraints again in order to solve our original problem. Our algorithm proceeds in two stages. First, we compute for each $m \in A$ the maximal profit that the realization of m can possibly yield. Then, we use the maximal profit of each edge to compute a conventional alignment.

Lemma 1. *The relaxed ILP can be solved in $\mathcal{O}(|A|^2)$ time.*

Suppose the variable $x_m = 0$, then due to (4) all $y_{lm} = 0$ as well. For $x_m = 1$, however, the optimal choice for all $m \in A$ is given by

$$\begin{aligned} \max \quad & \sum_{l \in A} w_{lm} y_{lm} + w_m \\ & \sum_{l \in I} x_l \leq 1 && \forall I \in \mathcal{I} \\ & \sum_{l \in A} y_{lm} \leq 1 \\ & x, y \geq 0 && \text{integer} \end{aligned}$$

To put it differently: For each $m \in A$ we compute the maximal profit that the alignment edge can possibly realize. The maximum profit consists of its own weight w_m plus the best interaction match that can be realized, if m is part of the solution. Let p_m be the maximum profit of alignment edge m and let \hat{y}_{lm} be the realized interaction match.

In the second step, we compute the optimal overall profit by solving

$$\begin{aligned} \max \quad & \sum_{m \in A} p_m x_m \\ & \sum_{l \in I} x_m \leq 1 && \forall I \in \mathcal{I} \\ & x \geq 0 && \text{integer} \end{aligned}$$

Let \bar{x} be the solution to the alignment problem above. We claim that the solution of the relaxed problem is given by $\bar{y}_{lm} = \hat{y}_{lm}\bar{x}_m$ for all $l, m \in A$.

Assume that (\bar{y}_{lm}, \bar{x}) is not the optimal solution to the relaxed problem, but another solution that does not realize the maximal profit. Then, according to the objective function, there exist $y_{vw} = 0$ and/or $x_w = 0$ that would realize a higher score. There are two possibilities: (a) $y_{vw} = 0$ and $x_w = 1$ (with $y_{xw} = 1$ being the interaction match chosen to be realized by alignment edge w): This implies that $w_{vw} > w_{xw}$. In this case, however, $y_{vw} = 1$ holds and not $y_{xw} = 1$, since the interaction match with the highest score is chosen in the first step of the algorithm. Therefore, there cannot be an $y_{vw} = 0$ whose realization would yield a higher score. (b) $x_w = 0$: This implies that for an element c from the set I (remember that pairs of elements of the same set I are crossing each other) holds $p_c > p_w$, because otherwise w would have been realized. Again, there cannot be an $x_w = 0$ yielding a higher score in case of $x_w = 1$.

For analyzing the running time of the entire algorithm, two things have to be taken into account: First, choosing the interaction match that realizes the maximal profit. Second, computing an alignment, given the single profits p . If the set of all interaction matches that could possibly be realized by alignment edge m is being computed in a preprocessing phase, selecting the best possible interaction match can be accomplished in constant time by means of priority queues (regard the weight of the interaction matches as the priority, then extracting the element with the highest priority can be done in constant time).

Computing the alignment dominates the overall running time and can be done in $\mathcal{O}(|A|^2)$ with the Needleman-Wunsch algorithm. □

4 Lagrangian Relaxation

We solve the original problem by moving the dropped constraints (3) into the objective function and by penalizing its violation. We assign a Lagrangian multiplier λ to the constraint. The task is then to find Lagrangian multipliers that provide the best bound to the original problem. The Lagrangian problem is given by:

$$\begin{aligned}
 \max \quad & \sum_{m \in A} \sum_{l \in A} w_{lm} y_{lm} + \sum_{m \in A} w_m x_m + \sum_{l \in A} \sum_{m \in A, l < m} \lambda_{lm} (y_{lm} - y_{ml}) \\
 & \sum_{l \in I} x_l \leq 1 \qquad \qquad \qquad \forall I \in \mathcal{I} \\
 & \sum_{l \in A} y_{lm} \leq x_m \qquad \qquad \qquad \forall m \in A \\
 & x, y \geq 0 \qquad \qquad \qquad \text{integer}
 \end{aligned}$$

By setting $\lambda_{ml} = -\lambda_{lm}$ for $l < m$ and λ_{ll} to 0, the objective function can be rewritten as

$$\max \sum_{m \in A} \sum_{l \in A} (\lambda_{lm} + w_{lm}) y_{lm} + \sum_{m \in A} w_m x_m \tag{6}$$

A common and highly efficient way to compute optimal (or near-optimal) Lagrangian multipliers is by employing iterative subgradient optimization. First, compute the subgradients within the i -th iteration:

$$s_{lm}^i = \bar{y}_{lm} - \bar{y}_{ml} \text{ for all } l, m \in A, l < m .$$

Then, using these subgradients, a series of λ_{lm}^i with $i = 1, \dots, n$ is given by

$$\lambda_{lm}^{i+1} = \begin{cases} \lambda_{lm}^i & \text{if } s_{lm}^i = 0 \\ \max(\lambda_{lm}^i - \gamma_i, -w_{lm}) & \text{if } s_{lm}^i = 1 \\ \min(\lambda_{lm}^i + \gamma_i, w_{lm}) & \text{if } s_{lm}^i = -1 \end{cases} \quad (7)$$

and $\lambda_{lm}^0 = 0$ for all $m, l \in A$

The speed of convergence to the optimal value of the Lagrangian relaxed problem depends heavily on the proper choice of the step size γ . A fundamental result [14] states that for $\lim_{i \rightarrow \infty} \gamma_i = 0$ and $\sum_i^\infty \gamma_i = \infty$ the value always converges to the optimal value. The harmonic series $\gamma_i = \frac{1}{i}$ with $i = 1, \dots, \infty$ satisfies both conditions; it is not being used in practice, however, since the speed of convergence turns out to be rather poor.

Another well-known formula, given in [9] and also applied by Lancia [4], computes the step size γ_i of the i -th iteration by

$$\gamma_i = \mu \frac{\text{UB} - \text{LB}}{\sum_{m,l \in A} (s_{lm}^i)^2}$$

where UB and LB denote the best upper and lower bound computed so far, μ is a common parameter used within subgradient optimization and s_{lm}^i are the subgradients evaluated in the i -th iteration.

5 Computing Feasible Solutions

The formula for computing the series of γ_i contains the term LB, the best lower bound found so far. In the following, we describe how to derive such a lower bound given a traditional sequence alignment and a set of interaction edges.

Since the set of alignment edges in the alignment \mathcal{A} is given, we only need to compute the best possible set of interaction edges such that no pair is in conflict.

It turns out that this problem can be reduced to the **Maximum Weighted Independent Set** problem. Consider the edges of \mathcal{A} as nodes and every pair of interaction edges (i_1, i_2) whose endpoints are adjacent to a pair $(l, m) \in \mathcal{A} \times \mathcal{A}$ as the edges of the graph. We call the resulting graph the **Conflict Graph** with edge weights given by the sum of the corresponding interaction edges in G . Figure 2 shows an alignment (dashed lines are interaction edges) with the corresponding interaction matching graph.

Lemma 2. *The maximum weight independent set of the Conflict Graph corresponds to the maximum weight interaction matching of the alignment.*

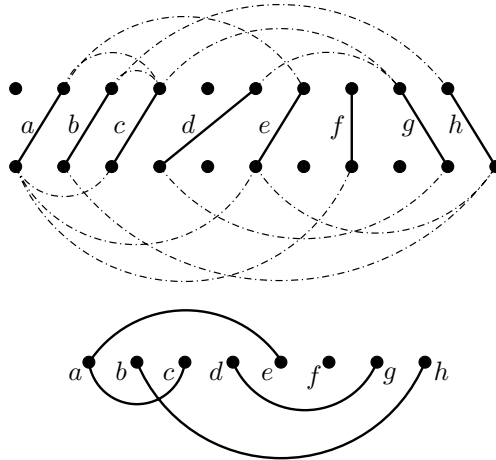


Fig. 2. Alignment and interaction matching graph

Recall the definition of a matching of maximum weight. In a matching every node is incident to at most one edge, the sum of the weights of the matching edges is maximal. It is not hard to see that the two properties for a structural alignment are satisfied:

1. Every node in the original graph is incident to (at most) one interaction edge (remember that nodes in the interaction matching graph are in one-to-one correspondence to alignment edges of the original graph).
2. The sum of weights of the interaction edges is maximal. □

It is trivial to recover the realized interaction edges from the solution of the matching problem, since each edge in the interaction matching graph corresponds to an interaction match.

6 Computational Results

We implemented our algorithm in C++ using the LEDA library [1] and used a 2.8 GHz Pentium IV with 2 GB of main memory for the computations.

We adapted the approach of Reinert et al. [11]. For generating the alignment edges we are starting from a conventional sequence alignment with affine gap costs (gap open and gap extension penalty are set to 6 and 2, respectively) and insert all alignment edges realized by any suboptimal alignment scoring better than a fixed threshold s below the optimal score. Matches and mismatches have a score of 4 and 1, one single interaction match counts 8.

In order to evaluate our approach we made two different kinds of experiments. First, we test our implementation with the same data set

Table 1. Comparison Lagrange vs. Branch-and-Cut

Inst.	Branch-and-Cut	Lagrange	Inst.	Branch-and-Cut	Lagrange
1	12563	12609	9	11975	12034
2	11566	11611	10	12055	12141
3	11744	11814	11	11618	11649
4	12260	12298	12	11611	11692
5	11709	11734	13	11491	11572
6	11569	11719	14	11521	11605
7	12193	12263	15	12067	12101
8	11586	11752	16	11804	11863

as used in [11]. Reinert *et al.* could align 23S ribosomal RNA sequences of length 1400 with suboptimality 3 (≈ 1500 -1600 alignment edges) in 2 minutes, but with suboptimality 10 (≈ 2000 -2100 alignment edges) they already needed over one hour. Table 1 shows the structural alignment scores that we compute for the same instances with suboptimality 50 (≈ 16000 -21000 alignment edges) in only about 10 minutes which is a significant improvement.

Secondly, to test whether our approach can preserve pseudoknots, we use 18 rRNA sequences from *Saccharomyces cerevisiae* (GenBank: M21017) and human (GenBank: K03432) having 1870 and 1995 bases, respectively. The structure files were obtained from [3]. There are three parts within the sequence that form pseudoknots: (a) bases 4-6 fold with 16-20 and 12-14 with 1198-1200, (b) 601-603 with 620-622 and 6 bases between 608-618 with bases from 623-635, and (c) bases 666-667 fold with 1145-1146 and parts of the sequence from 1090-1142 with some bases ranging from 1152-1161.

A traditional sequence alignment with affine gap costs was not able to realize the entire conserved structures, that is, some base pairs forming the pseudoknots are not realized. Our algorithm, however, aligned all base pairs forming the pseudoknots. To be precise: the traditional sequence alignment has a structural score of 12031, realizing 366 interaction matches, whereas the Lagrange method computes a score of 12662, realizing 409 interaction matches. Not only could we solve this alignment including several pseudoknots, but the program needed only 15 minutes and very moderate memory (≈ 300 MB).

Despite the very good computational results we have to remark that in the majority of problem instances—especially instances of 23S rRNA sequences—we could observe gaps between the lower and the upper bound: these gaps range from 2 to several hundred and therefore these instances were not solved to provable optimality (although the comparison with exact techniques like Branch-and-Cut shows that the scores are optimal or near-optimal). Our approach, however, could be easily integrated into a Branch-and-Bound framework to obtain provable optimal solutions.

7 Conclusion

In this paper we gave an ILP formulation of the RNA structural alignment problem and presented an efficient way to solve the problem by means of Lagrangian relaxation in $\mathcal{O}(|A|^2)$, A being the number of alignment edges. Our computational results show that we are able to compute structural alignments with a higher score within much shorter time than previous algorithms. Another advantage of our method is that, unlike most of the dynamic programming formulations, arbitrary pseudoknots can be handled in our framework and therefore the algorithm is able to detect conserved structures containing pseudoknots (as demonstrated in Sect. 6). Furthermore, the approach can be combined with other combinatorial optimization techniques, e.g., branch-and-bound or column generation. Besides this, we plan to extend the method to multiple sequence alignment and to integrate different gap scoring schemes.

Last but not least, our formulation is easy to implement (the actual algorithm is just a few hundred lines of code) and does not depend on third-party software like commercial optimization libraries.

References

1. Algorithmic Solutions. *The LEDA User Manual Version 4.5*, 2004. <http://www.algorithmic-solutions.com>.
2. V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In Z. Galil and E. Ukkonen, editors, *Proc. of the 6th Annual Symp. on Combinatorial Pattern Matching*, number 937 in LNCS, pages 1–16. Springer, 1995.
3. J. J. Cannone, S. Subramanian, M. N. Schnare, J. R. Collett, L. M. D’Souza, Y. Du, B. Feng, N. Lin, L. V. Madabusi, K. M. Muller, N. Pande, Z. Shang, N. Yu, and R. R. Gutell. The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC Bioinformatics*, 3(1), 2002.
4. A. Caprara and G. Lancia. Structural Alignment of Large-Size Proteins via Lagrangian Relaxation. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, pages 100–108. ACM Press, 2002.
5. F. Corpet and B. Michot. RNAAlign program: alignment of RNA sequences using both primary and secondary structures. *CABIOS*, 10(4):389–399, 1994.
6. S. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994.
7. P. A. Evans. Finding common subsequences with arcs and pseudoknots. In M. Crochemore and M. Patterson, editors, *Proc. Combinatorial Pattern Matching (CPM’99)*, volume 1645 of LNCS, pages 270–280. Springer, 1999.
8. J. Gorodkin, L. J. Heyer, and G. D. Stormo. Finding the most significant common sequence and structure motifs in a set of RNA sequences. *Nucl. Acids Res.*, 25:3724–3732, 1997.
9. M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.
10. I. L. Hofacker, S. H. F. Bernhart, and P. F. Stadler. Alignment of RNA base pairing probability matrices. *Bioinformatics*, 2004. In press.

11. H.-P. Lenhof, K. Reinert, and M. Vingron. A Polyhedral Approach to RNA Sequence Structure Alignment. *Journal of Comp. Biology*, 5(3):517–530, 1998.
12. D. H. Mathews and D. H. Turner. Dynalign: An algorithm for finding secondary structures common to two RNA sequences. *J. Mol. Biol.*, 317:191–203, 2002.
13. J. S. McCaskill. The Equilibrium Partition Function and Base Pair Binding Probabilities for RNA Secondary Structure. *Biopolymers*, 29:1105–1119, 1990.
14. B. Poljak. A general method of solving extremum problems. *Soviet Mathematics Doklady*, 8:593–597, 1967.
15. D. Sankoff. Simultaneous solution of the RNA folding, alignment, and proto-sequence problems. *SIAM J. Appl. Math.*, 45:810–825, 1985.
16. M. Waterman. Consensus methods for folding single-stranded nucleic acids. *Mathematical Methods for DNA Sequences*, pages 185–224, 1989.
17. J. Wuyts, Y. Van de Peer, T. Winkelmans, and R. De Wachter. The European database on small subunit ribosomal RNA. *Nucleic Acids Res*, 30:183–185, 2002.

Poly-APX- and PTAS-Completeness in Standard and Differential Approximation (Extended Abstract)

Cristina Bazgan, Bruno Escoffier, and Vangelis Th. Paschos

LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny,
75775 Paris Cedex 16, France
{bazgan, escoffier, paschos}@lamsade.dauphine.fr

Abstract. We first prove the existence of natural **Poly-APX**-complete problems, for both standard and differential approximation paradigms, under already defined and studied suitable approximation preserving reductions. Next, we devise new approximation preserving reductions, called **FT** and **DFT**, respectively, and prove that, under these reductions, natural problems are **PTAS**-complete, always for both standard and differential approximation paradigms. To our knowledge, no natural problem was known to be **PTAS**-complete and no problem was known to be **Poly-APX**-complete until now. We also deal with the existence of intermediate problems under **FT**- and **DFT**-reductions and we show that such problems exist provided that there exist **NPO**-intermediate problems under Turing-reduction. Finally, we show that **MIN COLORING** is **APX**-complete for the differential approximation.

1 Introduction

Many **NP**-complete problems are decision versions of natural optimization problems. Since, unless $\mathbf{P} = \mathbf{NP}$, such problems cannot be solved in polynomial time, a major question is to find polynomial algorithms producing solutions “close to the optimum” (in some pre-specified sense). Here, we deal with polynomial approximation of **NPO** problems (see [1] for a formal definition), i.e., of optimization problems the decision versions of which are in **NP**. As usual, we deal with problems the solution-values (or objective values) of which are integer.

For a problem Π in **NPO**, we distinguish three different versions of it: in the constructive version denoted also by Π , the goal is to determine the best solution y^* of an instance x ; in the evaluation version Π_e , we are only interested in determining the value of y^* ; finally, the decision version Π_d is as dealt in [2].

A polynomial approximation algorithm A for an optimization problem Π is a polynomial time algorithm that produces, for any instance x of Π , a feasible solution $y = A(x)$. The quality of y is estimated by computing the so-called approximation ratio. Two approximation ratios are commonly used in order to evaluate the approximation capacity of an algorithm: the standard ratio and the differential ratio. Given an instance x of an optimization problem Π , let $\text{opt}(x)$

be the value of an optimal solution, and $\omega(x)$ be the value of a worst feasible solution. This value is the optimal value of the same optimization problem (with respect to the set of instances and the set of feasible solutions for any instance) defined with the opposite objective (minimize instead of maximize, and vice-versa) with respect to Π . For a feasible solution y of x , denote by $m(x, y)$ its value. The *relative error* of y is defined as $r(x, y) = m(x, y)/\text{opt}(x)$. The *relative error* of y is defined as $\delta(x, y) = |m(x, y) - \omega(x)|/|\text{opt}(x) - \omega(x)|$. Following the above, standard approximation ratios for minimization problems are greater than, or equal to, 1, while for maximization problems these ratios are smaller than, or equal to 1. On the other hand, differential approximation ratio is always at most 1 for any problem.

By means of approximation ratios, **NPO** problems are classified with respect to their approximability properties. Particularly interesting approximation classes are, for the standard approximation paradigm, the classes **Poly-APX** (the class of the problems approximated within a ratio that is a polynomial, or the inverse of a polynomial when dealing with maximization problems, on the size of the instance), **APX** (the class of constant-approximable problems), **PTAS** (the class of problems admitting polynomial time approximation schemata) and **FPTAS** (the class of problems admitting fully polynomial time approximation schemata). We are referred to [1] for formal definitions. Analogous classes can be defined under the differential approximation paradigm: **Poly-DAPX**, **DAPX**, **DPTAS** and **DFPTAS**, are the differential counterparts of **Poly-APX**, **APX**, **PTAS** and **FPTAS**, respectively. Note that **FPTAS** \subsetneq **PTAS** \subsetneq **APX** \subsetneq **Poly-APX**, and **DFPTAS** \subsetneq **DPTAS** \subsetneq **DAPX** \subsetneq **Poly-DAPX**; these inclusions are strict unless **P** = **NP**.

During last two decades, several approximation preserving reductions have been introduced and, using them, hardness results in several approximability classes have been studied. We quote here four approximation preserving reductions that are central to our paper: **PTAS**, **DPTAS**, **F** and **E** (see also [3] for short definitions of them).

The **P**-reduction defined in [4] and extended in [5, 6] (been renamed **PTAS**-reduction) allows existence of **APX**-complete problems as **MAX INDEPENDENT SET- B** , or **MIN METRIC TSP**, etc (see [1, 2] for formal definitions about **NPO** problems mentioned in the paper).

In differential approximation, analogous results have been obtained in [7] under **DPTAS**-reduction. Natural problems such as **MAX INDEPENDENT SET- B** , or **MIN VERTEX COVER- B** are shown to be **DAPX**-complete.

Under **F**-reduction ([4]), only one (not very natural) problem (derived from **MAX VARIABLE-WEIGHTED SAT**) is known to be **PTAS**-complete. **DPTAS**-completeness has been done until now, but in any case **F**-reduction does not allow it.

Finally, the **E**-reduction ([8]) allows existence of **Poly-APX-PB**-complete problems but the existence of **Poly-APX**-complete problems has been left open.

An **NPO** problem Π is *approximable* if and only if there exists a polynomial q such that, for any instance x and for any feasible solution

$y \in \text{Sol}(x)$, $m(x, y) \leq q(|x|)$. It is said **polynomially bounded** if and only if there exists a polynomial q such that, for any instance x , $|\text{opt}(x) - \omega(x)| \leq q(|x|)$. The notion of diameter boundness is very useful and intuitive when dealing with the differential approximation paradigm. The class of polynomially bounded **NPO** problems will be denoted by **NPO-PB**, while the class of diameter polynomially bounded **NPO** problems will be denoted by **NPO-DPB**. Analogously, for any (standard or differential) approximation class **C**, we will denote by **C-PB** (resp., **C-DPB**) the subclass of polynomially bounded (resp., diameter polynomially bounded) problems of **C**.

The main results of this paper deal with the existence of complete problems for **Poly-APX**, **Poly-DAPX**, **FPTAS** and **DFPTAS**. **Poly-APX**-completeness is shown via PTAS-reduction ([6]), while **Poly-DAPX**-completeness is shown via DPTAS-reduction ([7, 9]). We define two new reductions, called FT and DFT, respectively, and show that, using them, natural problems as MAX PLANAR INDEPENDENT SET, MIN PLANAR VERTEX COVER, or BIN PACKING are complete for **PTAS** (the two first ones), or for **DPTAS** (all the three). Next, we study the existence of intermediate¹ problems for these reductions. We prove that such problems exist provided that there exist intermediate problems in **NPO** under the seminal Turing-reduction (see [1] for its definition). Finally, we prove that MIN COLORING is **DAPX**-complete under DPTAS-reduction. This is the first problem that is **DAPX**-complete but not **APX**-complete.

Results are given here without detailed proofs which can be found in [3].

2 Poly-APX-Completeness

As mentioned in [8], the nature of the E-reduction does not allow transformation of a non-polynomially bounded problem into a polynomially bounded one. In order to extend completeness in the whole **Poly-APX** we have to use a larger (less restrictive) reduction than E. In what follows, we show that PTAS-reduction can do it. Before continuing, we need the following notions defined in [8].

A problem $\Pi \in \text{NPO}$ is said **operator \oplus -closed** if and only if there exist an operator \oplus and a function f , both computable in polynomial time, such that:

- \oplus associates with any pair $(x_1, x_2) \in \mathcal{I}_\Pi \times \mathcal{I}_\Pi$ an instance $x_1 \oplus x_2 \in \mathcal{I}_\Pi$ with $\text{opt}(x_1 \oplus x_2) = \text{opt}(x_1) + \text{opt}(x_2)$;
- with any solution $y \in \text{sol}_\Pi(x_1 \oplus x_2)$, f associates two solutions $y_1 \in \text{sol}_\Pi(x_1)$ and $y_2 \in \text{sol}_\Pi(x_2)$ such that $m(x_1 \oplus x_2, y) = m(x_1, y_1) + m(x_2, y_2)$.

Let **Poly** be the set of functions from \mathbb{N} to \mathbb{N} bounded by a polynomial. A function $F : \mathbb{N} \rightarrow \mathbb{N}$ is said **operator \oplus -closed** for **Poly** if and only if for any $f \in \text{Poly}$, there exist three constants k , c and n_0 such that, for any $n \geq n_0$, $f(n) \leq kF(n^c)$.

¹ For two complexity classes \mathbf{C}_1 and \mathbf{C}_2 , $\mathbf{C}_1 \subseteq \mathbf{C}_2$, and a reduction R preserving membership in \mathbf{C}_1 , a problem is called **\mathbf{C}_2 -intermediate**, if it is neither **\mathbf{C}_2 -complete** under R , nor it belongs to \mathbf{C}_1 .

A maximization problem $\Pi \in \mathbf{NPO}$ is $\mathbf{Poly-APX}$ and only if there exist a transformation T from 3SAT to Π , two constants n_0 and c and a function F , hard for **Poly**, such that, given an instance x of 3SAT on $n \geq n_0$ variables and a number $N \geq n^c$, instance $x' = T(x, N)$ belongs to \mathcal{I}_Π and verifies the following properties:

1. if x is satisfiable, then $\text{opt}(x') = N$, otherwise $\text{opt}(x') = N/F(N)$;
2. given a solution $y \in \text{sol}_\Pi(x')$ such that $m(x', y) > N/F(N)$, one can polynomially determine a truth assignment satisfying x .

Note that, since 3SAT is **NP**-complete, a problem Π is canonically hard for **Poly-APX**, if any decision problem $\Pi' \in \mathbf{NP}$ reduces to Π along Items 1 and 2 just above.

Theorem 1. $\Pi \in \mathbf{NPO}$ is $\mathbf{Poly-APX}$ if and only if Π is $\mathbf{Poly-APX}$ PTAS.

(\Rightarrow) Let Π' be a maximization problem of **Poly-APX** and let \mathbf{A} be an approximation algorithm for Π achieving approximation ratio $1/c(\cdot)$, where $c \in \mathbf{Poly}$ (the case of minimization will be dealt later). Let Π be an additive problem, canonically hard for **Poly-APX**, let F be a function hard for **Poly** and let k and c' be such that (for $n \geq n_0$, for a certain value n_0) $nc(n) \leq k(F(n^{c'}) - 1)$. Let, finally, $x \in \mathcal{I}_{\Pi'}$, $\varepsilon \in]0, 1[$ and $n = |x|$. Set $m = m(x, \mathbf{A}(x))$; then $m \geq \text{opt}_{\Pi'}(x)/c(n)$. We uniformly partition the interval $[0, mc(n)]$ of possible values for $\text{opt}_{\Pi'}(x)$ into $q(n) = 2c(n)/\varepsilon$ sub-intervals (remark that q is a polynomial). Consider, for $i \in \{1, \dots, q(n)\}$, the set of instances $\mathcal{I}_i = \{x : \text{opt}_{\Pi'}(x) \geq imc(n)/q(n)\}$.

Set $N = n^{c'}$. We construct, for any i , an instance χ_i of Π such that: if $x \in \mathcal{I}_i$, then $\text{opt}_\Pi(\chi_i) = N$, otherwise, $\text{opt}_\Pi(\chi_i) = N/F(N)$. We define $f(x, \varepsilon) = \chi = \bigoplus_{1 \leq i \leq q(n)} \chi_i$. Observe that $c(n)/q(n) = \varepsilon/2$.

Let y be a solution of χ and let j be the largest i for which $m(\chi_i, y_i) > N/F(N)$, where y_i is the track of y on χ_i . Then, one can compute a solution ψ' of x such that $m(x, \psi') \geq jm\varepsilon/2$. We define

$$\psi = g(x, y, \varepsilon) = \text{argmax} \{m(x, \psi'), m(x, \mathbf{A}(x))\}$$

Note that $m(x, \psi) \geq \max\{m, jm\varepsilon/2\}$.

We show in [3] that $r(x, \psi) \geq r(\chi, y)(1 - (3\varepsilon/4))$, i.e., reduction just sketched is a PTAS-reduction with $c(\varepsilon) = \varepsilon/(4 - 3\varepsilon)$. ■

For the case where the problem Π' (in the proof of Theorem 1) is a minimization problem, one can reduce it to a maximization problem (for instance using the E-reduction of [8], p. 12) and then one can use the reduction of Theorem 1. Since the composition of an E- and a PTAS-reduction is a PTAS-reduction, the result of Theorem 1 applies also for minimization problems.

Combination of Theorem 1, of remark just above and of the fact that MAX INDEPENDENT SET is $\mathbf{Poly-APX}$ ([8]), produces the following concluding theorem.

Theorem 2. MAX INDEPENDENT SET *Poly-APX* *PTAS*

3 Poly-APX-Completeness Under the Differential Paradigm

The fact that function f (instance-transformation) of DPTAS-reduction ([7]) is multi-valued allows us to relax the constraint that a **Poly-DAPX**-complete problem has to be additive; we simply impose that it is canonically hard for **Poly-APX**.

Theorem 3. (\dots) , $\Pi \in \mathbf{NPO}$ *Poly-APX* *Poly-DAPX* *DPTAS* Π

(\dots) Let Π be canonically hard for **Poly-APX**, for some function F hard for **Poly**, let $\Pi' \in \mathbf{Poly-DAPX}$ be a maximization problem and let \mathbf{A} be an approximation algorithm for Π' achieving differential approximation ratio $1/c(\cdot)$, where $c \in \mathbf{Poly}$. Finally, let x be an instance of Π' of size n .

We will use the central idea of [7] (see also [9] for more details). We define a set $\Pi'_{i,l}$ of problems derived from Π' . For any pair (i, l) , $\Pi'_{i,l}$ has the same set of instances and the same solution-set as Π' ; for any instance x and any solution y of x , set $m_{i,l}(x, y) = \max\{0, \lfloor m(x, y)/2^i \rfloor - l\}$. Considering x as instance of any of the problems $\Pi'_{i,l}$, we will build an instance $\chi_{i,l}$ of Π , obtaining so a multi-valued function f . Our central objective is, informally, to determine a set of pairs (i, l) such that we will be able to build a “good” solution for Π' using “good” solutions of $\chi_{i,l}$.

Let $\varepsilon \in]0, 1[$; set $M_\varepsilon = 1 + \lfloor 2/\varepsilon \rfloor$ and let c' and k be such that (for $n \geq n_0$ for some n_0) $nc(n) \leq kF(nc')$ (both c' and k may depend on ε). Assume finally, without loss of generality, that $n \geq k$ and set $N = nc'$. Then, $1/F(N) \leq 1/c(n)$. Set $m = m(x, \mathbf{A}(x))$. In [7], a set \mathcal{F} of pairs (i, l) is built such that: $|\mathcal{F}|$ is polynomial with n and, furthermore, there exists a pair (i_0, l_0) in \mathcal{F} such that:

$$\delta_{i_0, l_0}(x, y) \geq 1 - \varepsilon \implies \delta(x, y) \geq 1 - 3\varepsilon \tag{1}$$

$$\text{opt}_{i_0, l_0}(x, y) \leq M_\varepsilon \tag{2}$$

Let q be an integer. Consider, for any pair $(i, l) \in \mathcal{F}$, the set of instances $\mathcal{I}_{i,l}^q = \{x \in \mathcal{I}_{\Pi'_{i,l}} : \text{opt}_{i,l}(x) \geq q\}$. More precisely, consider these instance-sets for $q \in \{0, \dots, M_\varepsilon\}$. For any pair $(i, l) \in \mathcal{F}$ and for any $q \in \{0, \dots, M_\varepsilon\}$, one can build an instance $\chi_{i,l}^q$ of Π such that: $\text{opt}_\Pi(\chi_{i,l}^q) = N$ if $\text{opt}_{i,l}(x) \geq q$, $\text{opt}_\Pi(\chi_{i,l}^q) = N/F(N)$, otherwise. We set $f : f(x, \varepsilon) = (\chi_{i,l}^q, (i, l) \in \mathcal{F}, q \in \{0, \dots, M_\varepsilon\})$.

Let $y = (y_{i,l}^q, (i, l) \in \mathcal{F}, q \in \{0, \dots, M_\varepsilon\})$ be a solution of $f(x, \varepsilon)$. Set $L_y = \{(i, l, q) : m(\chi_{i,l}^q, y_{i,l}^q) > N/F(N)\}$. For each $(i, l, q) \in L_y$, one can determine a solution $\psi_{i,l}^q$ of x (seen as instance of $\Pi'_{i,l}$) with value at least q . Set $\psi = g(x, y, \varepsilon) = \text{argmax}\{m(x, \mathbf{A}(x)), m(x, \psi_{i,l}^q), (i, l, q) \in L_y\}$.

Consider now a pair (i_0, l_0) verifying (1) and (2) and set $q_0 = \text{opt}_{i_0, l_0}(x)$. We can show ([3]) that if $\delta(x_{i_0, l_0}^{q_0}, y_{i_0, l_0}^{q_0}) \geq 1 - 3\varepsilon$, then $\delta(x, \psi) \geq 1 - 3\varepsilon$. Considering $\varepsilon' = 3\varepsilon$ and $c(\varepsilon') = \varepsilon'$, the reduction just sketched is a DPTAS-reduction. ■

Using the fact that MAX INDEPENDENT SET is canonically hard for **Poly-APX**, Theorem 3 directly exhibits the existence of a **Poly-DAPX**-complete problem.

Theorem 4. MAX INDEPENDENT SET *Poly-DAPX* *DPTAS*

Note that we could obtain the **Poly-DAPX**-completeness of canonically hard problems for **Poly-APX** even if we forbade DPTAS-reduction to be multi-valued. However, in this case, we should assume (as in Section 2) that Π is additive (and the proof of Theorem 3 would be much longer).

4 PTAS-Completeness

In order to study **PTAS**-completeness, we introduce a new reduction, called FT-reduction, preserving membership in **FPTAS**.

Let Π and Π' be two **NPO** maximization problems. Let $\square_\alpha^{\Pi'}$ be an oracle for Π' producing, for any $\alpha \in]0, 1]$ and for any instance x' of Π' , a feasible solution $\square_\alpha^{\Pi'}(x')$ of x' that is an $(1 - \alpha)$ -approximation for the standard ratio.

Definition 1. Π FT Π' ($\square_\alpha^{\Pi'}$, $\Pi \leq_{\text{FT}} \Pi'$) $\mathbf{A}_\varepsilon(x, \square_\alpha^{\Pi'})$

- $x \in \Pi, \mathbf{A}_\varepsilon \dots (1 - \varepsilon)$
- $\square_\alpha^{\Pi'}(x') \dots |x'| \dots 1/\alpha, \mathbf{A}_\varepsilon \dots |x| \dots 1/\varepsilon$

For the case where at least one among Π and Π' is a minimization problem it suffices to replace $1 - \varepsilon$ or/and $1 - \alpha$ by $1 + \varepsilon$ or/and $1 + \alpha$, respectively.

Clearly, FT-reduction transforms a fully polynomial time approximation schema for Π' into a fully polynomial time approximation schema for Π , i.e., it preserves membership in **FPTAS**.

The F-reduction is a special case of FT-reduction since the latter explicitly allows multiple calls to oracle \square . Also, FT-reduction seems allowing more freedom in the way Π is transformed into Π' ; for instance, in F-reduction, function g transforms an optimal solution for Π' into an optimal solution for Π , i.e., F-reduction preserves optimality; this is not the case for FT-reduction. This freedom will allow us to reduce non polynomially bounded **NPO** problems to **NPO-PB** ones. In fact, it seems that FT-reduction is larger than F. This remains to be confirmed. Such proof is not trivial and is not tackled here.

In what follows, given a class $\mathbf{C} \subseteq \mathbf{NPO}$ and a reduction R , we denote by $\overline{\mathbf{C}}^R$ the closure of \mathbf{C} under R , i.e., the set of problems in \mathbf{NPO} that R -reduce to some problem in \mathbf{C} .

The basic result of this section (Theorem 5) follows immediately from Lemmata 1 and 2. Lemma 1 introduces a property of Turing-reduction for \mathbf{NP} -hard problems. In Lemma 2, we transform (under certain conditions) a Turing-reduction into a FT-reduction. Proofs of the two lemmata are given for maximization problems. The case of minimization is completely analogous.

Lemma 1. *If $\Pi \in \overline{\mathbf{NPO}}^R$ and $\Pi' \in \mathbf{NP}$, then $\Pi' \in \overline{\mathbf{NPO}}^R$.*

Let Π be an \mathbf{NPO} problem and q be a polynomial such that $|y| \leq q(|x|)$, for any instance x of Π and for any feasible solution y of x . Assume that encoding $n(y)$ of y is binary. Then $0 \leq n(y) \leq 2^{q(|x|)} - 1$. We consider the following problem $\hat{\Pi}$ (see also [5]) which is the same as Π up to its objective function that is defined by $m_{\hat{\Pi}}(x, y) = 2^{q(|x|)+1}m_{\Pi}(x, y) + n(y)$.

Clearly, if $m_{\hat{\Pi}}(x, y_1) \geq m_{\hat{\Pi}}(x, y_2)$, then $m_{\Pi}(x, y_1) \geq m_{\Pi}(x, y_2)$. So, if y is an optimal solution for x (seen as instance of $\hat{\Pi}$), then it is also an optimal solution for x (seen, this time as instance of Π).

Remark now that for $\hat{\Pi}$, the evaluation problem $\hat{\Pi}_e$ and the constructive problem $\hat{\Pi}$ are equivalent. Indeed, given the value of an optimal solution y , one can determine $n(y)$ (hence y) by computing the remainder of the division of this value by $2^{q(|x|)+1}$.

Since Π' is \mathbf{NP} -hard, we can solve the evaluation problem $\hat{\Pi}_e$ if we can solve the (constructive) problem Π' . Indeed, we can solve $\hat{\Pi}_e$ using an oracle solving, by dichotomy, the decision version $\hat{\Pi}_d$ of $\hat{\Pi}$; $\hat{\Pi}_d$ reduces to the decision version Π'_d of Π' by a Karp-reduction (see [1, 2] for a formal definition of this reduction); finally, one can solve Π'_d using an oracle for the constructive problem Π' . So, with a polynomial number of queries to an oracle for Π' , one can solve both $\hat{\Pi}_e$ and $\hat{\Pi}$, and the proof of the lemma is complete. ■

We now show how, starting from a Turing-reduction (that only preserves optimality) between two \mathbf{NPO} problems Π and Π' where Π' is polynomially bounded, one can devise an FT-reduction transforming a fully polynomial time approximation schema for Π' into a fully polynomial time approximation schema for Π .

Lemma 2. *If $\Pi' \in \mathbf{NPO-PB}$ and $\Pi \in \overline{\mathbf{NPO}}^R$, then $\Pi \in \overline{\mathbf{NPO-PB}}^R$.*

Let Π be an \mathbf{NPO} problem and suppose that there exists a Turing-reduction between Π and Π' . Let $\square_{\alpha}^{\Pi'}$ be an oracle computing, for any instance x' of Π' and for any $\alpha > 0$, a feasible solution y' of x' such that $r(x', y') \geq 1 - \alpha$. Moreover, let p be a polynomial such that for any instance x' of Π' and for any feasible solution y' of x' , $m(x', y') \leq p(|x'|)$.

Let x be an instance of Π . The Turing-reduction claimed gives an algorithm solving Π using an oracle for Π' . Consider now this algorithm where we use, for any query to the oracle with the instance x' of Π' , the approximate oracle $\square_{\alpha}^{\Pi'}(x')$, with $\alpha = 1/(p(|x'|) + 1)$. This algorithm produces an optimal solution, since a solution y' being an $(1 - (1/(p(|x'|) + 1)))$ -approximation for x' is an optimal one (recall that we deal with problems having integer-valued objective functions). Indeed,

$$\begin{aligned} \frac{m_{\Pi'}(x', y')}{\text{opt}_{\Pi'}(x')} &\geq 1 - \frac{1}{p(|x'|) + 1} \implies m_{\Pi'}(x', y') > \text{opt}_{\Pi'}(x') - 1 \\ &\implies m_{\Pi'}(x', y') = \text{opt}_{\Pi'}(x') \end{aligned}$$

It is easy to see that this algorithm is polynomial when $\square_{\alpha}^{\Pi'}(x')$ is polynomial in $|x'|$ and in $1/\alpha$. Furthermore, since any optimal algorithm for Π can be a posteriori seen as a fully polynomial time approximation schema, we immediately conclude $\Pi \leq_{\text{FT}} \Pi'$ and the proof of the lemma is complete. ■

Combination of Lemmata 1 and 2, immediately derives the basic result of the section expressed by the following theorem.

Theorem 5. $\Pi' \in NP \implies \Pi \in NPO$ and $\Pi' \in NPO\text{-PB} \implies \Pi \in NPO$, $\Pi' \in FT \implies \Pi \in FT$

From Theorem 5, one can immediately deduce the two corollaries that follow.

Corollary 1. $\overline{PTAS}^{\text{FT}} = NPO$

Corollary 2. $\overline{PTAS} = PTAS$ and $\overline{FT} = FT$

For instance, MAX PLANAR INDEPENDENT SET and MIN PLANAR VERTEX COVER are in both **PTAS** ([10]) and **NPO-PB**. What has been discussed in this section concludes then the following result.

Theorem 6. MAX PLANAR INDEPENDENT SET and MIN PLANAR VERTEX COVER $\in PTAS$ and $\in FT$

Remark that the results of Theorem 6 cannot be trivially obtained using the F-reduction of [4].

5 DPTAS-Completeness

In order to study **DPTAS**-completeness we will again use a new reduction called **DFT**-reduction. Since it is very similar to the **FT**-reduction of Section 4 (up to consideration differential ratios instead of standard ones), its definition is omitted.

Let us note that one of the basic features of differential approximation ratio is that it is stable under affine transformations of the objective functions of the problems dealt. In this sense, problems for which the objective functions of the ones are affine transformations of the objective functions of the others are approximate equivalent for the differential approximation paradigm (this is absolutely not the case for standard paradigm). The most notorious case of such problems is the pair **MAX INDEPENDENT SET** and **MIN VERTEX COVER**. Affine transformation is nothing else than a very simple kind of differential-approximation preserving reduction, denoted by **AF**, in what follows. Two problems Π and Π' are affine equivalent if $\Pi \leq_{\text{AF}} \Pi'$ and $\Pi' \leq_{\text{AF}} \Pi$. Obviously affine transformation is both a **DPTAS**- and a **DFT**-reduction (as this latter one is derived from Definition 1).

Results of this section are derived analogously to the case of the **PTAS**-completeness of Section 4: we show that any **NP**-hard problem, that belongs to both **NPO-DPB** and **DPTAS**, is **DPTAS**-complete. The basic result of this paragraph (Theorem 7) is an immediate consequence of Lemma 1 and of the following Lemma 3, differential counterpart of Lemma 2 (see [3] for the proof).

Lemma 3. $\Pi' \in \text{NPO-DPB} \xrightarrow{\text{DFT}} \text{NPO} \xrightarrow{\text{DFT}} \Pi'$

Theorem 7. $\Pi' \in \text{NPO-DPB} \xrightarrow{\text{DFT}} \text{NP} \xrightarrow{\text{DFT}} \Pi'$

Corollary 3. $\overline{\text{DPTAS}}^{\text{DFT}} = \text{NPO}$

Corollary 4. $\text{NPO-DPB} \xrightarrow{\text{DFT}} \text{DPTAS} \xrightarrow{\text{DFT}} \text{DPTAS}$

The following concluding theorem deals with the existence of **DPTAS**-complete problems.

Theorem 8. **MAX PLANAR INDEPENDENT SET, MIN PLANAR VERTEX COVER, BIN PACKING** $\xrightarrow{\text{DFT}}$ **DPTAS** $\xrightarrow{\text{DFT}}$ **DPTAS**

For **DPTAS**-completeness of **MAX PLANAR INDEPENDENT SET**, just observe that, for any instance G , $\omega(G) = 0$. So, standard and differential approximation ratios coincide for this problem; moreover, it is in both **NPO-PB** and **NPO-DPB**. Then, the inclusion of **MAX PLANAR INDEPENDENT SET** in **PTAS** suffices to conclude its membership in **DPTAS** and, by Corollary 4, its **DPTAS**-completeness.

MAX PLANAR INDEPENDENT SET and **MIN PLANAR VERTEX COVER** are affine equivalent; hence, the former **AF**-reduces to the latter. Since **AF**-reduction is a particular kind of **DFT**-reduction, the **DPTAS**-completeness of **MIN PLANAR VERTEX COVER** is immediately concluded.

Finally, since **BIN PACKING** \in **DPTAS** ([11]) and also **BIN PACKING** belongs to **NPO-DPB**, its **DPTAS**-completeness immediately follows. ■

6 About Intermediate Problems Under FT- and DFT-Reductions

FT-reduction is weaker than the F-reduction of [4]. Furthermore, as mentioned before, this last reduction allows existence of **PTAS**-intermediate problems. The question of existence of such problems can be posed for FT-reduction too. In this section, we handle it via the following theorem.

Theorem 9.
$$\begin{array}{ccc} \text{NPO} & & \\ \text{PTAS} & \text{FT} & \end{array}$$

([3]) Let $\Pi \in \mathbf{NPO}$ be intermediate for the Turing-reduction. Suppose that Π is a maximization problem (the minimization case is completely similar). Let p be a polynomial such that, for any instance x and any feasible solution y of x , $m(x, y) \leq 2^{q(|x|)}$. Consider the following maximization problem $\tilde{\Pi}$ where:

- instances are the pairs (x, k) with x an instance of Π and k an integer in $\{0, \dots, 2^{q(|x|)}\}$;
- for an instance (x, k) of $\tilde{\Pi}$, its feasible solutions are the feasible solutions of the instance x of Π ;
- the objective function of $\tilde{\Pi}$ is:

$$m_{\tilde{\Pi}}((x, k), y) = \begin{cases} |x, k| & \text{if } m(x, y) \geq k \\ |x, k| - 1 & \text{otherwise} \end{cases}$$

It suffices now to show the three following properties:

1. $\tilde{\Pi} \in \mathbf{PTAS}$;
2. if $\tilde{\Pi}$ were in **FPTAS**, then Π would be polynomial;
3. if $\tilde{\Pi}$ were **PTAS**-complete, then Π would be **NPO**.

Obviously, if Properties 1, 2 and 3 hold ([3]), then the theorem is concluded since their combination deduces that if Π is **NPO**, then $\tilde{\Pi}$ is **PTAS**-intermediate, under FT. ■

We now state an analogous result about the existence of **DPTAS**-intermediate problems under DFT-reduction.

Theorem 10.
$$\begin{array}{ccc} \text{NPO} & & \\ \text{DPTAS} & \text{DFT} & \end{array}$$

The proof is analogous to one of Theorem 9, up to modification of definition of $\tilde{\Pi}$ (otherwise, $\tilde{\Pi} \notin \mathbf{DPTAS}$, because the value of the worst solution of an instance (x, k) is $|x, k| - 1$). We only have to add, for any instance (x, k) of $\tilde{\Pi}$, a new feasible solution y_x^0 with value $m_{\tilde{\Pi}}((x, k), y_x^0) = 0$. Then, the result claimed is got in exactly the same way as in the proof of Theorem 9. ■

² We emphasize this expression in order to avoid confusion with usual **NPO**-completeness considered under the strict-reduction ([12]).

7 A New DAPX-Complete Problem Not APX-Complete

All **DAPX**-complete problems given in [7] are also **APX**-complete under the **E**-reduction ([8]). An interesting question is if there exist **DAPX**-complete problems that are not also **APX**-complete for some standard-approximation preserving reduction. In this section, we positively answer this question by the following theorem.

Theorem 11. $\text{MIN COLORING} \leq_{\text{DAPX}} \text{MAX UNUSED COLORS} \leq_{\text{DPTAS}} \text{MIN COLORING}$

Consider problem **MAX UNUSED COLORS**. For this problem, standard and differential approximation ratios coincide and coincide also with differential ratio of **MIN COLORING**. So, $\text{MAX UNUSED COLORS} \leq_{\text{AF}} \text{MIN COLORING}$.

As proved in [13], **MAX UNUSED COLORS** is **MAX-SNP**-hard under **L**-reduction, a particular kind of **E**-reduction. Also, $\overline{\text{MAX-SNP}}^{\text{E}} = \text{APX-PB}$ ([8]). **MAX INDEPENDENT SET-B** belongs to **APX-PB**, so, **MAX INDEPENDENT SET-B** **E**-reduces to **MAX UNUSED COLORS**. **E**-reduction is a particular kind of **PTAS**-reduction, so, $\text{MAX INDEPENDENT SET-B} \leq_{\text{PTAS}} \text{MAX UNUSED COLORS}$.

Standard and differential approximation ratios for **MAX INDEPENDENT SET-B**, on the one hand, standard and differential approximation ratios for **MAX UNUSED COLORS**, and differential ratio of **MIN COLORING**, on the other hand, coincide. So, $\text{MAX INDEPENDENT SET-B} \leq_{\text{DPTAS}} \text{MIN COLORING}$.

DPTAS- and **AF**-reductions just exhibited, together with the fact that their composition is obviously a **DPTAS**-reduction, establish immediately the **DAPX**-completeness of **MIN COLORING**. ■

As we have already mentioned, **MIN COLORING** is, until now, the only problem known to be **DAPX**-complete but not **APX**-complete. In fact, in standard approximation paradigm, it belongs to the class **Poly-APX** and is inapproximable, in a graph of order n , within $n^{1-\varepsilon}$, $\forall \varepsilon > 0$, unless **NP** coincides with the class of problems that could be optimally solved by slightly super-polynomial algorithms ([14]).

8 Conclusion

We have defined suitable reductions and obtained natural complete problems for important approximability classes, namely, **Poly-APX**, **Poly-DAPX**, **PTAS** and **DPTAS**. Such problems did not exist until now. This work extends also the ones in [7, 9] further specifying and completing a structure for differential approximability. The only among the most notorious approximation classes for which we have not studied completeness is **Log-DAPX** (the one of the problems approximable within differential ratios of $O(1/\log|x|)$). This is because, until now, no natural **NPO** problem is known to be differentially approximable within inverse logarithmic ratio. Work about definition of **Log-DAPX**-hardness is in progress.

Another point that deserves further study, is the structure of approximability classes beyond **DAPX** that are defined not with respect to the size of the instance but to the size of other parameters as natural as $|x|$. For example, dealing with graph-problems, no research is conducted until now on something like **Δ -APX**-, or **Δ -DAPX**-completeness where Δ is the maximum degree of the input graph. Such works miss to both standard and differential approximation paradigms. For instance, a question we are currently trying to handle is if **MAX INDEPENDENT SET** is, under some reduction, **Δ -APX**-complete, or **Δ -DAPX**-complete. Such notion of completeness, should lead to achievement of inapproximability results (in terms of graph-degree) for several graph-problems.

Finally, the existence of natural **PTAS**-, or **DPTAS**-intermediate problems (as **BIN PACKING** for **APX** under **AP**-reduction) for **F**-, **FT**- and **DFT**-reductions remains open.

References

1. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and approximation. Combinatorial optimization problems and their approximability properties. Springer, Berlin (1999)
2. Garey, M.R., Johnson, D.S.: Computers and intractability. A guide to the theory of NP-completeness. W. H. Freeman, San Francisco (1979)
3. Bazgan, C., Escoffier, B., Paschos, V. Th.: Completeness in standard and differential approximation classes: Poly-(D)APX- and (D)PTAS-completeness. Cahier du LAMSADE 217, LAMSADE, Universit Paris-Dauphine (2004) Available on <http://www.lamsade.dauphine.fr/cahiers.html>.
4. Crescenzi, P., Panconesi, A.: Completeness in approximation classes. Information and Computation **93** (1991) 241–262
5. Ausiello, G., Crescenzi, P., Protasi, M.: Approximate solutions of NP optimization problems. Theoret. Comput. Sci. **150** (1995) 1–55
6. Crescenzi, P., Trevisan, L.: On approximation scheme preserving reducibility and its applications. Theory of Computing Systems **33** (2000) 1–16
7. Ausiello, G., Bazgan, C., Demange, M., Paschos, V. Th.: Completeness in differential approximation classes. In: Mathematical Foundations of Computer Science, MFCS'03. Number 2747 in Lecture Notes in Computer Science, Springer-Verlag (2003) 179–188
8. Khanna, S., Motwani, R., Sudan, M., Vazirani, U.: On syntactic versus computational views of approximability. SIAM J. Comput. **28** (1998) 164–191
9. Ausiello, G., Bazgan, C., Demange, M., Paschos, V. Th.: Completeness in differential approximation classes. Cahier du LAMSADE 204, LAMSADE, Universit Paris-Dauphine (2003) Available on <http://www.lamsade.dauphine.fr/cahiers.html>.
10. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. J. Assoc. Comput. Mach. **41** (1994) 153–180
11. Demange, M., Monnot, J., Paschos, V. Th.: Bridging gap between standard and differential polynomial approximation: the case of bin-packing. Appl. Math. Lett. **12** (1999) 127–133
12. Orponen, P., Mannila, H.: On approximation preserving reductions: complete problems and robust measures. Technical Report C-1987-28, Dept. of Computer Science, University of Helsinki, Finland (1987)

13. Halldórsson, M.M.: Approximating discrete collections via local improvements. In: Proc. Symposium on Discrete Algorithms, SODA. (1995) 160–169
14. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. In: Proc. Conference on Computational Complexity. (1996) 278–287

Efficient Algorithms for k Maximum Sums

Fredrik Bengtsson and Jingsen Chen

Department of Computer Science and Electrical Engineering,
Luleå University of Technology,
S-971 87 Luleå, Sweden

Abstract. We study the problem of computing the k maximum sum subsequences. Given a sequence of real numbers $\langle x_1, x_2, \dots, x_n \rangle$ and an integer parameter k , $1 \leq k \leq \frac{1}{2}n(n-1)$, the problem involves finding the k largest values of $\sum_{\ell=i}^j x_\ell$ for $1 \leq i \leq j \leq n$. The problem for fixed $k = 1$, also known as the maximum sum subsequence problem, has received much attention in the literature and is linear-time solvable. Recently, Bae and Takaoka presented a $\Theta(nk)$ -time algorithm for the k maximum sum subsequences problem. In this paper, we design efficient algorithms that solve the above problem in $O\left(\min\{k + n \log^2 n, n\sqrt{k}\}\right)$ time in the worst case. Our algorithm is optimal for $k \geq n \log^2 n$ and improves over the previously best known result for any value of the user-defined parameter k . Moreover, our results are also extended to the multi-dimensional versions of the k maximum sum subsequences problem; resulting in fast algorithms as well.

1 Introduction

One frequently used pedagogical example of optimizing computer programs for speed is [\[1\]](#). Given a sequence of real numbers $\langle x_1, x_2, \dots, x_n \rangle$, the objective is to find a subsequence of consecutive elements with the maximum sum among all such subsequences $\langle x_i, \dots, x_j \rangle$ for $1 \leq i \leq j \leq n$. This problem is a special one-dimensional version of [\[2\]](#) that serves as a maximum likelihood estimator for some patterns when processing digital pictures [\[2, 3\]](#). For the latter problem, a two-dimensional array of real numbers is given, and the task is to find a rectangular subarray with the largest possible sum among all such subarrays. The problem arises in data mining and graphics as well [\[4, 5\]](#).

The maximum sum subsequence problem has a linear time sequential solution [\[6, 7\]](#). For an $m \times n$ matrix of real numbers, the maximum subarray problem can be solved in $O(m^2n)$ time (assuming that $m \leq n$) [\[6, 7, 8\]](#). By reducing the problem to graph distance matrix multiplications, Tamaki and Tokuyama [\[9\]](#) present the first sub-cubic time algorithm for the maximum sum subarray problem. Following the same strategy, Takaoka [\[10\]](#) gives a modified algorithm with a slightly better time complexity. In the context of parallel computations, optimal speed-up algorithms on different models of parallel computations have been developed [\[8, 11, 12, 13\]](#).

A natural extension of the above problems is to compute the k largest sums over all possible subsequences/subarrays. As mentioned above, the maximum sum subsequence problem can be solved in $\Theta(n)$ time [6, 7]. Closer inspection reveals that a similar approach (by scanning the input array and updating the maximum sum subsequence encountered so far) seemingly will not work for our generalized problem (since the number of candidate subsequences becomes $\Theta(n^2)$). Recently, a $\Theta(nk)$ -time algorithm for computing the k maximum sum subsequences has been presented in [14]. In this paper, we will develop faster algorithms to solve these problems. Our results improve over the previously best known algorithms.

In the next section, the computational problems to be solved are defined and the main results of this paper are described. Efficient algorithms for solving the k maximum sum subsequences problem are developed in Section 3. For small values of the parameter k , we show in Section 4 how to solve the problem even faster. An extension of our results to higher dimensions are presented in Section 5. Finally, we conclude the paper with some open problems.

2 Problem Settings and Main Results

Given a sequence X of real numbers $\langle x_1, x_2, \dots, x_n \rangle$ and an integer k , $1 \leq k \leq \frac{1}{2}n(n-1)$, the k maximum sum subsequence problem is to select k pairs of indices $\{(i_\ell, j_\ell) : 1 \leq i_\ell \leq j_\ell, \ell = 1, 2, \dots, k\}$ such that the (range) sums $\sum_{p=i_\ell}^{j_\ell} x_p$, $\ell = 1, 2, \dots, k$, are the k largest values among all the possible sums $\sum_{\ell=i}^j x_\ell$ for $1 \leq i \leq j \leq n$. Notice that we do not require that the sums are sorted. The algorithms presented in this paper compute only the range sums without pointing out the subsequences explicitly; extending the algorithms to output the subsequences computed as well is straightforward.

The k maximum sum subsequences problem becomes trivial if $k = \Theta(n^2)$: In this case, an optimal algorithm for the problem runs in $\Theta(n^2)$ time in the worst case. To achieve this, first compute all possible sums $\sum_{\ell=i}^j x_\ell$ ($1 \leq i \leq j \leq n$) and then select the k^{th} largest sum. Then, traverse the list of sums and pick all sums larger than or equal to the k^{th} sum in order to get all the k largest sums. For arbitrary value of the parameter k , we first propose an algorithm running in $O(k + n \log^2 n)$ time in the worst case. Then, we show that the problem can also be solved in $O(n\sqrt{k})$ time; which is even faster for small values of the user-specified parameter k . Combining these two algorithms results in a worst-case running time $O(\min\{k + n \log^2 n, n\sqrt{k}\})$. This time complexity is the best possible for $k \geq n \log^2 n$. Previously, the optimal methods were known only for two extreme cases: $k = O(1)$ and $k = \Theta(n^2)$. Notice that the previously best known result for this problem is $\Theta(nk)$ [14] for $1 \leq k \leq \frac{1}{2}n(n-1)$. Our algorithms are faster for any value of k .

The 2-dimensional version of the k maximum sum subsequence problem is as follows: Given two-dimensional array of order $m \times n$, find k orthogonal continuous subregions that has a sum at least as large as the k^{th} largest

sum. The sums are chosen from the set of all continuous subarrays of the given array. A straightforward solution would cost $\Theta(m^2n^2)$ via an enumeration of all possible sums. We will show how to reduce the time complexity for this problem to $O(\min\{m^2C, m^2n^2\})$ in the worst case in Section 5, where $C = \min\{k + n \log^2 n, n\sqrt{k}\}$. Extending the algorithms to higher dimensions results in a $O(n^{2d-2} \min\{C, n^2\})$ -time solution for a given d -dimension array with size n in each dimension.

In processing our algorithms for the above problems, we need to perform the selection and the search operation in the $A + B$ of a sequence $A = \langle a_1, a_2, \dots, a_m \rangle$ and a sequence $B = \langle b_1, b_2, \dots, b_n \rangle$ of real numbers, where $A + B$ is the set $\{a_i + b_j | 1 \leq i \leq m \text{ and } 1 \leq j \leq n\}$; optimal algorithms have been developed for those problems [15, 16]. The set, $A \triangle B$, of all the good elements in $A + B$ is defined as $\{a_i + b_j | 1 \leq i \leq j \leq n\}$ assuming that $n = m$. The elements of $A + B$ that are not good will be referred to as bad elements.

Definition 1. Let x be a real number, $A = \langle a_1, a_2, \dots, a_m \rangle$ a sequence of real numbers, and $M = [a_{i,j}]_{n \times m}$ a matrix of real numbers. Then, $\text{rank}(x; A) \triangleq \|\{a | a \in A, a \leq x\}\|$ and $\text{rank}(x; M) \triangleq \|\{a_{i,j} | a_{i,j} \leq x, 1 \leq i \leq n, 1 \leq j \leq m\}\|$.

Definition 2. Let x be a real number, $M = [a_{i,j}]_{n \times n}$ a matrix of real numbers. Then, $\text{rank}_G(x; M)$ is the number of good elements in M , that is, $\|\{a_{i,j} | a_{i,j} \leq x, 1 \leq i \leq j \leq n\}\|$.

Throughout the paper we assume that either the input array is already resident in internal memory, or each element can be computed as needed in constant time. All logarithms are to the base 2.

3 K Maximum Sums Algorithm

In this section, we present an algorithm for solving the k maximum sum subsequences problem on a given sequence of length n . Our algorithm consists of five phases. In the first phase, the problem is reduced to finding the top k maximum values over all the good elements in some matrix of order $n \times n$. The second phase performs repeated constraint searches which decreases the number of candidate elements to $O(kn)$. A procedure of range reduction will be carried out in the third phase. This procedure is able to reduce the number of candidates further to $\Theta(k)$. In the fourth phase, a worst-case linear-time selection algorithm is done on the remaining candidates. The output of this phase is an element, x , that is the k^{th} largest range sum. The final phase involves finding the good elements whose values are not less than x .

3.1 Problem Transformation

Given an input instance $X = \langle x_1, \dots, x_n \rangle$ of the k maximum sum subsequences problem, we can construct the prefix-sum sequence $P = \langle p_1, \dots, p_n \rangle$ of X in $O(n)$ time in the worst case: $p_1 = x_1, p_{i+1} = p_i + x_{i+1}$ for $1 \leq i \leq n - 1$. Let Q

be the set $\{-p \mid p \in P\}$, where $-0 = 0$. It is clear that any range sum $\sum_{l=i}^j x_l$ of the sequence X is equal to $p_j - p_{i-1}$, for $i \leq j$. Hence, the goal of the k maximum sum subsequences problem becomes finding the k largest values among all the good elements in the Cartesian sum set $Q + P$. Although selection, search, and ranking problems in matrices are well studied, previous algorithms on matrices and Cartesian sum sets are not directly applicable due to the goodness of the elements. In the subsections that follow, we will solve the selection, search, and ranking problems in Cartesian sum sets, which are of their own interests. Given two sequences A and B , we assume for simplicity that all the elements in $A + B$ are distinct and our algorithms operate on the matrix M of the form $A + B$ (whenever such an interpretation is needed, observing that M is not actually constructed). We will describe our algorithms for the k smallest elements in the sets under consideration. It is straightforward to adapt the algorithms when the largest elements are requested.

3.2 Constraint Search and Ranking

Let $A = \langle a_1, a_2, \dots, a_n \rangle$ and $B = \langle b_1, b_2, \dots, b_n \rangle$ be two sequences of numbers. Consider the problem of computing the number of bad elements in $A + B$ that are less than or equal to a given number x . Namely, we are interested in calculating the value $\mathbf{rank}(x; A + B) - \mathbf{rank}_G(x; A + B)$. This rank computation can be done by a search in $A + B$ with respect to the good elements.

Notice that each column in the matrix $M = \hat{A} + B$ is sorted if \hat{A} is the sorted output when sorting A in non-decreasing order. We first perform a binary search on each column j to compute the number of elements less than x in that column. Let i_j be the largest index such that $M[i_j, j] \leq x$. That is, all elements in column j above i_j are smaller than or equal to x . Since we have not rearranged or reordered the sequence B , the n^{th} column of M contains no bad elements. Column $n - i$ will contain i bad elements for $1 \leq i \leq n - 1$. Observing that if column j has a bad element at row i , then every column ℓ , $1 \leq \ell \leq j - 1$, will have a bad element at row i as well. To locate the position of these bad elements with respect to positions $\{i_1, \dots, i_j\}$, we scan the matrix column by column from the right to the left and process one bad element at a new position in each column when going left.

For this purpose, let π be the permutation that sorts A ; i.e., $A[\pi(i)] = \hat{A}[i]$ for $i = 1, 2, \dots, n$. Now, construct an array, L , of length m to store the information about the positions and the number of bad elements when scanning. Initially, L is an array of zeros. Consider one column at a time. Upon processing columns $n, n - 1, \dots, j + 1$, we will assign 1 to $L[\pi(j)]$ if the current column j has a bad element at row $\pi(j)$ (that is, $\pi(j) < j$). Hence, L has value 1's at the positions corresponding to the rows where the elements are bad for the current column. Therefore, the number of bad elements in column j that is less than or equal to x is $n_j = \sum_{i=1}^{i_j} L[i]$. The only difference between column $j + 1$ and column j is that column j may contain one more bad element. As mentioned above, the location of this bad element can be computed with the help of the permutation π in constant time. Moreover, the computation of n_j can be done in $O(\log n)$

time using an algorithm for computing dynamic prefix sums [17]. The number of bad elements computed for each column could be accumulated as a real number over all columns. Since the number of columns is n and each column requires $O(\log n)$ time which gives a total time of $O(n \log n)$ for the whole matrix. The pseudo-code for our algorithm is as follows.

1. Initiate array L of length n with all zeros.
2. Let $l = 0$.
3. for $j = n - 1$ downto 1 do locate the position for x in column j by a standard binary search. Let i_j be the largest index such that $M[i_j, j] \leq x$.
4. for $j = n$ downto 1 do
 - (a) Compute $s = \sum_{i=1}^{i_j} L[i]$ using the prefix sums algorithm in [17].
 - (b) Let $l = l + s$.
 - (c) If $\pi(j) < j$, then let $L[\pi(j)] = 1$ and update the data structure by employing the algorithm in [17].
5. return l .

The above algorithm assumes that the sequence, A , is sorted. Since the length of A is n , the sorting takes $O(n \log n)$ time. We have proven the following:

Lemma 1. . . . $A = \langle a_1, a_2, \dots, a_n \rangle$. . . $B = \langle b_1, b_2, \dots, b_n \rangle$. . .
 $A + B$. . . x . . . $O(n \log n)$. . .

3.3 Range Reduction and Listing

The problem considered in this subsection is to find and list out all the good elements in $A + B$ that are less than or equal to a given number x , where $A = \langle a_1, a_2, \dots, a_n \rangle$ and $B = \langle b_1, b_2, \dots, b_n \rangle$ are two given sequences of numbers. When employing the result here in our solution to the k maximum sum subsequences problem, the range of the candidates is decreased and the procedure is repeated until the sub-problems can be solved in desired time bounds.

Similar to the previous subsection, consider the matrix $M = \hat{A} + B$ where \hat{A} is the sorted output when sorting A in non-decreasing order. In this case, each column in M is sorted. As discussed above, each column, except the rightmost one, will contain a number of bad elements. Moreover, column j can contain one and at most one more bad element than column $j + 1$. Again, this bad element can be found with the help of the sort permutation π of A . That is, $A[\pi(i)] = \hat{A}[i]$ for $i = 1, 2, \dots, n$.

Our algorithm uses two lists: A list L for storing all the good elements searched for and a doubly linked list, called . . . , in order to jump over all the bad elements during the search. It is important that the bad elements are not scanned, since there can be $\Theta(n^2)$ of them. We will implement the jump list with two arrays of indices, the . . . , F , and the . . . , G . Each element of the forward array is the index of the next good element in the

current column. Each element in the backward array is the index of the previous good element in the same column. The same forward and backward arrays will be used for each column, but updated appropriately.

Starting from the rightmost column, we will process the columns of the matrix as follows: Begin with the smallest element of the column, if it is less than or equal to x , add it to L and continue to process the next good element in the same column. If not, we are done with the column. Initially, $F[i] = i + 1$ and $G[i] = i - 1$. Before listing and storing the good elements ($\leq x$) in the current column, we compute the new bad element for this column (by using π). Notice that the number of such bad elements is at most one. Then we update F and G as follows: Let the index of the new bad element in the current column be i . Let $F[G[i]] = F[i]$ and $G[F[i]] = G[i]$. We search for good elements smaller than or equal to x in the order of $F[1], F[F[1]], F[F[F[1]]], \dots$. Assume that the number of good elements in column j whose values are less than or equal to x is r_j . Thus, the time used in processing column j will be r_j plus a constant amount of work (for F , G , and an extra element scanned), which is $O(r_j + 1)$. The total time required will, therefore, equal $O\left(\sum_{j=1}^n (r_j + 1)\right) = O(\text{rank}_G(x; A + B) + n)$. Next, we present the pseudo-code:

1. Initiate $F[i] = i + 1, 0 \leq i \leq n - 1$.
2. Initiate $G[i] = i - 1, 1 \leq i \leq n$.
3. Initiate empty list L .
4. for $j = n$ downto 1 do
 - (a) Let $i = F[0]$. while $M[i, j] \leq x$ do
 - i. Add $M[i, j]$ to L .
 - ii. Let $i = F[i]$.
 - (b) Let $i = \pi(j)$.
 - (c) Let $F[G[i]] = F[i]$ and $G[F[i]] = G[i]$.

Since sorting the sequence A takes $O(n \log n)$ time, we have

Lemma 2. $\dots A \dots B \dots n \dots$
 $\dots x, \dots A + B \dots$
 $\dots x \dots O(\text{rank}_G(x; A + B) + n \log n)$

3.4 Putting Things Together

Now, we are ready to present our algorithm for the k maximum sum subsequences problem. Given a sequence $X = \langle x_1, \dots, x_n \rangle$ of numbers and an integer $k \geq 1$.

1. If $k = \Omega(n^2)$, then enumerate all the possible range sums of X and select the $\min\{k, \frac{1}{2}n(n - 1)\}$ largest ones; Exit.
2. Compute the prefix sums sequence, P , of X : $P = \langle p_1, \dots, p_n \rangle$, where $p_{i+1} = p_i + x_{i+1}, i = 1, \dots, n - 1$ and $p_1 = x_1$.
3. Let $Q = -P = \{-p | p \in P\}$.

4. Sort Q in non-decreasing order; denote the sorted output by A .
Sort P in non-decreasing order; denote the sorted output by B .
5. $a \leftarrow 1$
 $b \leftarrow k$
6. Repeat
 - (a) Select the b^{th} largest element, x , in $A+B$.
 - (b) Compute the number, m , of the good elements in $Q + P$ that is $\geq x$.
 - (c) If $m < k$, then $a \leftarrow b$ and $b \leftarrow 2b$.
until $m \geq k$.
7. while $m > 2k$ do
 - (a) $x \leftarrow$ the $\lfloor \frac{a+b}{2} \rfloor^{\text{th}}$ largest element in $A + B$.
 - (b) $m \leftarrow$ the number of good elements $\geq x$ in $Q + P$.
 - (c) if $m < k$ then $a \leftarrow \lfloor \frac{a+b}{2} \rfloor$ else $b \leftarrow \lfloor \frac{a+b}{2} \rfloor$
8. Let S be the set of all good elements in $Q + P$ whose values are greater than or equal to x .
9. Find all the k largest elements in S .

After the problem transformation, our algorithm searches for an element, x , by repeated range reductions so that the relative rank of x in $Q + P$ is between k and $2k$. Next, the algorithm discards all the good elements whose values $\geq x$ and all the bad elements in $Q + P$. Then, the k maximum range sums of X are the first k largest values among the remaining elements.

3.5 Complexity Analysis

The above algorithm runs in $O(k + n \log^2 n)$ time in the worst case. In fact, Step 1 is done in $O(n^2) = O(k)$ time. Steps 2 and 3 runs in linear time. Step 4 takes $O(n \log n)$ time.

During each iteration of Step 6, (a) can be performed with $O(\sqrt{b}) = O(n)$ operations [16] since both A and B are sorted; (b) takes $O(n \log n)$ time with the help of the sorted sequence A of Q by Lemma 1. The number of iterations can easily be computed from the following lemma.

Lemma 3. Let x be an element in A . $N = \text{rank}(x; A) = r$,
 $r \leq \text{rank}(x; B) \leq N + r$, where $B = \{b|b \in A \text{ or } -b \in A\}$

Let all the elements in A whose values are less than or equal to x be a_1, a_2, \dots, a_r . Assume, without loss of generality, that all of a_1, \dots, a_i are strictly less than 0 and all of a_{i+1}, \dots, a_r are greater than or equal to 0, where $0 \leq i \leq r$. Obviously, $\text{rank}(x; B) \geq r$ for the set B defined in the lemma. On the other hand, the size of the following set: $\{b|b \in B \text{ or } b \leq x\}$, is at most $r + (r - i) + (N - r) = N + r - i \leq N + r$. The lemma follows. \square

Notice that in Step 6(c), the value of b is doubled each time. Therefore, when Step 6 is done, we have $a = 2^\ell k$ and $b = 2^{\ell+1}k$ for some integer $\ell \geq 0$. Let $N = \frac{1}{2}n(n - 1)$ and $r = k$. From the above lemma, we have $2^\ell k \leq N + k$ and thus, $\ell = O(\log \frac{N}{k}) = O(\log n)$. Hence, Step 6 takes at most $O(n \log^2 n)$ time.

Step 7 performs a binary-search-like operation, where each step does a selection and a computation of relative rank. Similar to that of Step 6, the time complexity of this step equals $O(n \log n \log(b - a)) = O(n \log^2 n)$. Step 8 requires $O(\text{rank}_G(x; Q + P)) = O(k)$ from Lemma 2 since we have already sorted the sequence Q before this step. Finally, Step 9 takes $O(k)$ time as well.

Theorem 1. *Let X be a sequence of n real numbers. Then the k largest range sums of X can be found in $O(k + n \log^2 n)$ time.*

4 Fast Computation When k Is Small

An obvious lower bound for the k maximum sum subsequences problem is $\Omega(k + n)$. Thus, the algorithm presented in the previous section gives the best possible running time when $k = \Omega(n \log^2 n)$, but it does not give the fastest algorithm for small values of k . Observing that if the input sequence is divided into two blocks of consecutive elements, all the subsequences having the k largest range sums must be entirely located in the left block, in the right block, or cross the border between the left and the right block. By crossing the border we mean that the left endpoint of the subsequence is located in the left block and the right endpoint in the right block.

In order to efficiently find the range sums crossing the border between the blocks, we will precompute all the suffix sums S of the left block and all the prefix sums P of the right block. After that, all the range sums crossing the border are now elements in the Cartesian sum of $S + P$. The k largest elements in $S + P$ can be found fast if we first sort the sets S and P , respectively. In fact, our algorithm produces not only the k maximum sum subsequences, but also the k largest prefix sums, the k largest suffix sums (in sorted order), and the sum of the given sequence. That is, our algorithm indeed solves the following problem:

- Input: A sequence X of n reals and an integer k , $1 \leq k \leq \frac{1}{2}n(n - 1)$.
- Output: K_X : the set of the k largest range sums of X
- P_X : the set of the $\min\{k, n\}$ largest prefix sums of X
- S_X : the set of the $\min\{k, n\}$ largest suffix sums of X
- $w(X)$: the sum of X

Notice first that if $k = \Theta(n^2)$, the above problem can be solved by first computing all the possible range sums of X and then select the k largest ones. This can be done in $\Theta(n^2)$ time in the worst case. Similarly, $w(X)$, P_X and S_X can be computed in $O(n \log n) = O(n^2)$ time. Therefore, we have

Lemma 4. *Let X be a sequence of n real numbers. Then the k largest range sums of X can be found in $\Theta(n^2)$ time when $k = \Theta(n^2)$.*

The above observations lead us to a divide-and-conquer approach to solving the given problem. However, in order to speed up the computation, the recursion will be forced to stop when the size of the underlying block drops below $\Theta(\sqrt{k})$. In that case, we will apply the method from the above lemma to that block.

AlgoMaxSum(X, k)

Input: A sequence X of n real numbers and an integer $k \geq 1$

Output: $(K_X, P_X, S_X, w(X))$

1. If $k = \Omega(n^2)$ then let $k' = \min\{k, \frac{1}{2}n(n-1)\}$, compute (K, P, S, w) according to Lemma 4 with the parameter integer k' , and exit.
2. Divide X into two blocks of equal size: $L = X[1..n/2]$ and $R = X[n/2+1..n]$.
3. Recursively solve the problem on L and R , respectively, and let
 - $(K_L, P_L, S_L, w(L)) \leftarrow \text{AlgoMaxSum}(L, k)$,
 - $(K_R, P_R, S_R, w(R)) \leftarrow \text{AlgoMaxSum}(R, k)$.
4. Find the set, C , of all the largest elements in $S_L + P_R$.
5. Select the k largest elements in $K_L \cup C \cup K_R$ resulting in the set K_X .
6. Assign the k largest elements in $P_L \cup (P_R + \{w_L\})$ to P_X .
7. Assign the k largest elements in $(S_L + \{w_R\}) \cup S_R$ to S_X .
8. Compute $w(X) = w(L) + w(R)$.

The correctness of the algorithm follows directly from the above description and the observation at the beginning of this section. We will analyze the time complexity step by step. Let $T(n, k)$ be the worst-case running time of the algorithm $\text{AlgoMaxSum}(X, k)$. In the case when $k = \Omega(n^2)$, the algorithm (now Step 1) costs $O(n^2)$ time. Step 2 takes $O(1)$ time and Step 3 requires $2T(n/2, k)$ time. Step 4 can be done by first finding the k^{th} largest element, x , in $S_L + P_R$ and then searching for all the elements in $S_L + P_R$ that are larger than x . Here we may assume without loss of generality that all the elements in $S_L + P_R$ are distinct. Notice that both S_L and P_R are sorted. Thus, x can be found in $O(k)$ time [15] and the search procedure above can be solved in $\Theta(k)$ time [16]. Step 5 takes $\Theta(k)$ time using any standard worst-case linear time selection algorithm. One way to implement Step 6 is to first merge P_L and $P_R + \{w_L\}$ and then choose the k largest elements in the resulting sorted set, which can be done in $\Theta(k)$ time. Similarly, Step 7 runs in $\Theta(k)$ time. The last step needs only constant time. To sum up, we have the following recurrence

$$T(n, k) = \begin{cases} 2T\left(\frac{n}{2}, k\right) + O(k) & \text{, if } k = o(n^2) \\ O(n^2) & \text{, if } k = \Omega(n^2) \end{cases}$$

With the substitution method, we have $T(n, k) = 2^i T\left(\frac{n}{2^i}, k\right) + O\left(\sum_{j=0}^{i-1} k2^j\right)$.

Letting $(n/2^i)^2 = k$ results in $2^i = n/\sqrt{k}$. Hence, $T(n, k) = (n/\sqrt{k}) T(\sqrt{k}, k) + O(n\sqrt{k}) = (n/\sqrt{k}) O(k) + O(n\sqrt{k}) = O(n\sqrt{k})$. Therefore,

Theorem 2. *Let k be an integer such that $k = o(n^2)$. Then the running time of $\text{AlgoMaxSum}(X, k)$ is $O(n\sqrt{k})$.*

Combining Theorems 1 and 2, we have

Corollary 1. *Let n be a positive integer and k be a positive integer such that $k = o(n^2)$. Then the running time of $\text{AlgoMaxSum}(X, k)$ is $O(\min\{k + n \log^2 n, n\sqrt{k}\})$.*

In the next section, we will show how to solve the high-dimensional problems incrementally by using our fast algorithms developed in Sections 3 and 4.

5 Higher Dimensional Cases

We first study the two-dimensional version of the problem; that is, to find all the orthogonal continuous subregions whose sums are at least as large as the k^{th} largest sum among all continuous subarrays of a given two-dimensional array. The following algorithm simply reduces the dimension parameter in the problem. Actually, for a given array $X = [x_{i,j}]_{1 \leq i \leq m, 1 \leq j \leq n}$ of real numbers, we transform the problem into $\Theta(m^2)$ one-dimensional k maximum sum subsequences problems, and the later problems are solved using our one-dimensional algorithms.

Algorithm: **AlgoMaxSum2D**(X, k)

Input: A two-dimensional array $X = [x_{i,j}]_{1 \leq i \leq m, 1 \leq j \leq n}$ of real numbers and an integer $k \geq 1$.

Output: The set K_X of the k largest sums among all possible sums $\sum_{i_1=j_1}^{j_2} \sum_{i_2=j_3}^{j_4} x_{i_1, i_2}$ for $1 \leq j_1 \leq j_2 \leq m, 1 \leq j_3 \leq j_4 \leq n$

1. Compute a new array, $Y = [y_{i,j}]$ of order $m \times n$, where $y_{i,j} = \sum_{l=1}^j x_{i,l}$.
2. For each i and j , $1 \leq i \leq j \leq m$,
 - (a) Create an array $A_{i,j} = \langle a_1, \dots, a_n \rangle$ such that $a_l = y_{j,l} - y_{i-1,l}$ for $l = 1, 2, \dots, n$.
 - (b) Solve the k maximum sum subsequences problem on $A_{i,j}$ with the parameter k ; output $K_{A_{i,j}}$.
3. Let K_X be the k largest elements in the union of $K_{A_{i,j}}$ for $1 \leq i \leq j \leq m$.

Clearly, this algorithm computes the k largest range sums. Notice that Step 1 actually computes the prefix sums for each column of X ; which can be done in $\Theta(mn)$ time. Step 2 requires $O(C)$ operations for every fixed i and j from Corollary 1, where $C = \min\{k + n \log^2 n, n\sqrt{k}\}$. Since the total number of index combinations is $\Theta(m^2)$, the time complexity of this step is thus $\Theta(m^2C)$. Note that the size of the union of $K_{A_{i,j}}$, $1 \leq i \leq j \leq m$, is $\Theta(m^2k)$. Hence, the selection (Step 3) requires $\Theta(m^2k)$ time in such a set. Therefore, the algorithm **AlgoMaxSum2D** takes $O(m^2C + m^2k)$ time. If $k = o(n^2)$, then $m^2C + m^2k = O(m^2n\sqrt{k})$. If $k = \Omega(n^2)$, however, we can run a straightforward algorithm running in $O(m^2n^2)$ time that enumerates all possible range sums and selects the k largest ones, instead of **AlgoMaxSum2D**. Therefore, we have

Theorem 3. Let $X = [x_{i,j}]_{1 \leq i \leq m, 1 \leq j \leq n}$ be an $m \times n$ array of real numbers. Then the k largest range sums of X can be found in $O(\min\{m^2C, m^2n^2\})$ time, where $C = \min\{k + n \log^2 n, n\sqrt{k}\}$.

An algorithm for the k maximum sum subarrays problem with a running time of $\Theta(m^2nk)$ has been presented in [14]. Notice that the value of the parameter k is between 1 and $\Theta(m^2n^2)$. Our algorithm improves over the above result for every value of k .

It is possible to extend our approach to the d -dimensional version of the maximum sum subarrays problem. Given a d -dimensional array of real numbers, with size n in each dimension, we can solve the k maximum sum subarrays problem in $O(n^{2d-2} \min\{C, n^2\})$ time; the details can be found in [18].

6 Conclusions

We have addressed the problem of computing k maximum sum subsequences/subarrays and proposed efficient algorithms. The bounds that we obtain improve substantially on previous results and our algorithms are optimal for some non-constant values of the parameter k ($k \geq n \log^2 n$). Previously, only for two extreme cases when $k = O(1)$ and $k = \Theta(n^2)$, the problem investigated could be solved optimally. We have made some progress in designing optimal algorithms. However, any algorithm that computes the k largest range sums requires at least $\Omega(n+k)$ operations in the worst case. It is an interesting problem to see whether this lower bound is achievable. Moreover, some new ideas will be needed in order to handle the higher-dimensional problems optimally.

References

1. Weiss, M.: Data Structures and Algorithm Analysis (2nd Edition). Addison-Wesley (1995)
2. Bentley, J.: Programming pearls: Algorithm design techniques. Communications of the ACM **27** (1985) 865–871
3. Grenander, U.: Pattern Analysis. Springer-Verlag, New York (1978)
4. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. (1993) 207–216
5. Fukuda, T., Morimoto, Y., Morishita, S., Tokuyama, T.: Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. (1996) 13–23
6. Bentley, J.: Programming pearls: Perspective on performance. Communications of the ACM **27** (1985) 1087–1092
7. Gries, D.: A note on the standard strategy for developing loop invariants and loops. Science of Computer Programming **2** (1982) 207–214
8. Smith, D.: Applications of a strategy for designing divide-and-conquer algorithms. Science of Computer Programming **8** (1987) 213–229
9. Tamaki, H., Tokuyama, T.: Algorithms for the maximum subarray problem based on matrix multiplication. In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. (1998) 446–452
10. Takaoka, T.: Efficient algorithms for the maximum subarray problem by distance matrix multiplication. In: Proceedings of the 2002 Australian Theory Symposium. (2002) 189–198
11. Akl, S., Guenther, G.: Application of broadcasting with selective reduction to the maximal sum subsegment problem. International Journal of High Speed Computing **3** (1991) 107–119

12. Perumalla, K., Deo, N.: Parallel algorithms for maximum subsequence and maximum subarray. *Parallel Processing Letters* **5** (1995) 367–373
13. Qiu, K., Akl, S.: Parallel maximum sum algorithms on interconnection networks. Technical Report No. 99-431, Jodrey School of Computer Science, Acadia University, Canada (1999)
14. Bae, S.E., Takaoka, T.: Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem. In: Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks. (2004) 247–253
15. Frederickson, G., Johnson, D.: The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *Journal of Computer and System Sciences* **24** (1982) 197–208
16. Frederickson, G., Johnson, D.: Generalized selection and ranking: Sorted matrices. *SIAM Journal on Computing* **13** (1984) 14–30
17. Riedewald, M., Agrawal, D., Abbadi, A.E.: Flexible data cubes for online aggregation. In: International Conference on Database Theory. LNCS (2001) 159–173
18. Bengtsson, F., Chen, J.: Computing the k maximum subarrays fast. Technical Report No. 2004:07, Luleå University of Technology, Luleå, Sweden (2004)

Equipartitions of Measures by 2-Fans

Sergey Bereg

Department of Computer Science,
University of Texas at Dallas, Box 830688,
Richardson, TX 75083, USA
besp@utdallas.edu,
<http://www.utdallas.edu/~besp>

Abstract. We study the problem of computing an equitable 2-fan for three masses distributed on the 2-sphere. The existence of an equitable 2-fan was shown by Bárány and Matoušek [3]. The contribution of this paper is two-fold. (i) We prove the existence of an infinite set of equitable 2-fans. (ii) We present an efficient algorithm for finding an equitable 2-fan when the mass distributions are discrete, i.e. finite sets of points. Both (i) and (ii) can be easily extended to mass distributions in the plane instead of the sphere.

1 Introduction

Balanced partitions of sets of points and dissections of mass distributions are a fundamental topic in Combinatorial Geometry [23] and received attention recently [3, 4, 6, 7, 22]. An k -fan of a mass distribution (measure) μ defined on a space X is a finite collection $\mathcal{K} = \{K_1, \dots, K_q\}$ of measurable sets forming a partition (dissection) of X such that $\mu(K_i) = \mu(\mathbb{R}^d)/q$ for each $i = 1, \dots, q$. We assume that the space X is either the plane \mathbb{R}^2 or the unit 2-sphere. We are interested in the partitions by k -fans [3], see Fig. 1 for examples. Equipartitions find applications in computer science [19] (for example, geometric range searching), in statistics [5] (for example, regression depth) and in “practice” [1] (for example, cutting a cake).

... Kaneko and Kano [15] conjectured that, for any qn red and qm blue points ($q, n, m > 0$ are integers) in the plane in general position, there are q disjoint convex polygons with n red and m blue points in each of them. The conjecture of Kaneko and Kano has been independently proven by Bespamyatnikh ... [6], Ito ... [14] (the case $q = 3$), Sakai [20]. If $q = 3$, the partition is a 3-fan with convex wedges.

Bárány and Matoušek [3] established a number of results concerning the existence (and non-existence) of balanced k -fans. In particular, they proved the existence of a 2-fan equipartitioning three masses on the sphere and a 3-fan equipartitioning two masses on the sphere.

Equipartitioning by a k -fan is related to the Ham Sandwich Theorem [2, 23]. The Ham Sandwich problem is well studied from the algorithmic point of

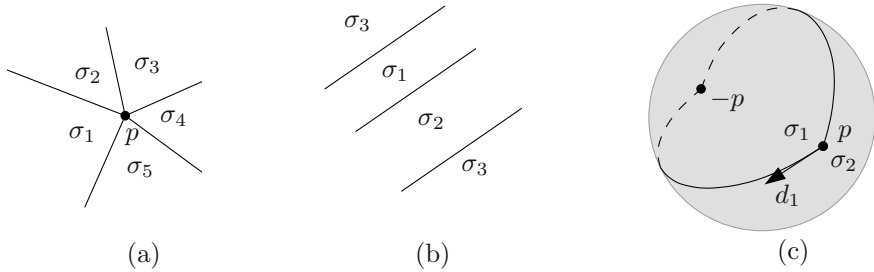


Fig. 1. (a) 5-fan in the plane with the center p , (b) 3-fan in the plane, and (c) 2-fan on the sphere with the center p

view [10, 12, 13, 17, 21, 24]. An optimal algorithm of Lo et al. [17] finds a Ham Sandwich cut in linear time. Very recently, Bose [7] studied the problem of finding a geodesic ham-sandwich cut which can be viewed as a generalization of the classical one.

We consider the problem of finding an equitable 2-fan for three discrete masses defined as finite sets of points. The brute-force approach based on the existence of 2-fan [3] is to check all possible 2-fans. There are $O(n^5)$ combinatorially different 2-fans for a set of n points. It would be useful for designing an efficient algorithm to prove the existence of an equitable 2-fan that satisfies some property. Mikio Kano conjectured [16] that every line contains the center of an equitable 2-fan. In this paper we prove the conjecture.

Theorem 1.

- (i) \dots
- (ii) \dots

We prove Theorem 1 in a slightly stronger form. We assume that points on a sphere are in \dots if (i) no three points lie on the same great circle, and (ii) no two points are \dots (i.e. their midpoint is the sphere center).

Theorem 2.

- (i) \dots
- (ii) \dots

This result can be stated in a more general setting: For any three finite Borel measures on the 2-sphere S , any Jordan curve on S whose ends are antipodal points contains the center of an equitable 2-fan.

The proof of Theorem 2 allows us to reduce the problem of finding an equitable 2-fan to the well-known problem of the slope selection [9]. The slope selection can be solved optimally in $O(n \log n)$ time using a deterministic algorithm [8, 9] and a randomized algorithm (and somewhat simpler) [11, 18]. We present an algorithm for finding an equitable 2-fan on the unit 2-sphere. The algorithm runs in $O(N \log^2 N)$ time where $2N$ is the total number of points.

2 Preliminaries

Let $S = \{p \mid p_x^2 + p_y^2 + p_z^2 = 1\}$ be the unit sphere in \mathbb{R}^3 . Let p be a point of S . For a direction d perpendicular to Op , we denote by $c(p, d)$ the great half-circle between p and $-p$ in the direction d . Let d_1 and d_2 be two distinct directions perpendicular to Op . The two great arcs $c(p, d_1)$ and $c(p, d_2)$ are considered to be a 2-fan $f = (p; d_1, d_2)$. They partition the sphere into two regions – denoted as $\sigma_1(f)$ and $\sigma_2(f)$, see Fig. 1 (c).

Suppose that $\lambda_1, \lambda_2, \lambda_3$ are absolutely continuous Borel probabilities on S such that all the measures are positive on non-void open sets (as in [4]). For a point p and a direction d_1 , there is a unique 2-fan $f(p; d_1, d_2)$ such that $\lambda_1(\sigma_1(f)) = 1/2$ (also there is a unique 2-fan such that $\lambda_1(\sigma_1(f)) + \lambda_2(\sigma_1(f)) + \lambda_3(\sigma_1(f)) = 3/2$). Let X be the set of all such 2-fans. We can associate a 2-frame (or just frame) with a 2-fan. Thus, the space of 2-fans is homeomorphic to Stiefel manifold $V_2(\mathbb{R}^3) \cong SO(3)$ of all orthonormal 2-frames in \mathbb{R}^3 .

From now on, we assume that the masses are discrete and colored: let R, B and G be sets of red, blue and green points on the sphere S , respectively. We assume that the set of each color contains an even number of points, namely $|R| = 2n, |B| = 2m$ and $|G| = 2k$. Let $P = R \cup B \cup G$ be the set of all given points and let $N = n + m + k$.

Let \mathcal{A} be the sphere arrangement generated by the great circles passing through pairs of points of P . Let p be a point from the interior of a face in \mathcal{A} and let d be a direction such that $c(p, d)$ avoids P . We define a 2-fan $f = (p; d, d')$ for the pair (p, d) so that the number of points in $P \cap \sigma_1(f)$ is N (note that the direction d' is not unique but the set $P \cap \sigma_1(f)$ is unique). For simplicity, we call the canonical 2-fan just 2-fan, denote it by $\phi(p, d)$, and denote its lunes by $\sigma_1(p, d)$ and $\sigma_2(p, d)$. We call p a vertex of the 2-fan and d a direction of the 2-fan.

Two 2-fans $\phi = (p, d)$ and $\phi' = (p, d')$ are called adjacent if the interior of one of the two lunes between $c(p, d)$ and $c(p, d')$ contains exactly one data point. Let $\sigma(p, d, d')$ denote the lune between $c(p, d)$ and $c(p, d')$ obtained by rotating $c(p, d)$ about the line passing through the origin O and p in clockwise order when we look from p towards O . A 2-fan $\phi' = (p, d')$ is called a neighbor of $\phi = (p, d)$ if exactly one data point lies in the interior of $\sigma(p, d, d')$. A 2-fan $\phi' = (p, d')$ is called a neighbor of $\phi = (p, d)$ if exactly one data point lies in the interior of $\sigma(p, d', d)$.

$$\text{Let } \delta(x) = \begin{cases} 0, & x = 0 \\ x/|x|, & x \neq 0 \end{cases} .$$

3 The Existence of an Equitable 2-Fan

To evaluate the quality of a 2-fan $(p, d) \in X$ we define a map $\mu : X \rightarrow \mathbb{Z}^2$ by $\mu(p, d) = (\mu_1(p, d), \mu_2(p, d))$ where $\mu_1(p, d)$ is the number of red points in $\sigma_1(p, d)$ and $\mu_2(p, d)$ is the number of blue points in $\sigma_1(p, d)$. For detecting if a 2-fan is

equitable we introduce a map $\alpha : X \rightarrow \mathbb{Z}^2$ by $\alpha(p, d) = (\alpha_1(p, d), \alpha_2(p, d))$ where $\alpha_1(p, d) = \delta(\mu_1(p, d) - n)$ and $\alpha_2(p, d) = \delta(\mu_2(p, d) - m)$. Clearly, a 2-fan (p, d) is equitable if and only if $\alpha(p, d) = (0, 0)$.

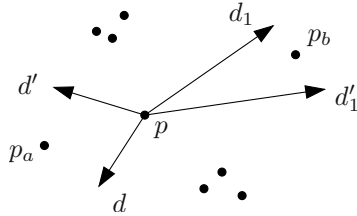


Fig. 2. The 2-fans $\phi = (p; d, d_1)$ and $\phi' = (p; d', d'_1)$

Lemma 1. $p \in \mathcal{A} \implies \exists f \in \mathcal{A} \text{ such that } \phi = (p, d)$
 $\phi' = (p, d') \implies \alpha(\phi) \neq 0 \implies \alpha(\phi') \neq 0 \implies \alpha(\phi)$
 $\alpha(\phi')$
 $()$

Without loss of generality we assume that ϕ' is the CW-neighbor of ϕ . Let $\mu(p, d) = (n_1, n_2)$ and $\mu(p, d') = (n'_1, n'_2)$. Let $\Sigma = \sigma_1(p, d) \cap P$ and $\Sigma' = \sigma_1(p, d') \cap P$. The sets Σ and Σ' differs by two points, say p_a and p_b , such that $\Sigma \cup \{p_b\} = \Sigma' \cup \{p_a\}$, see Fig. 2.

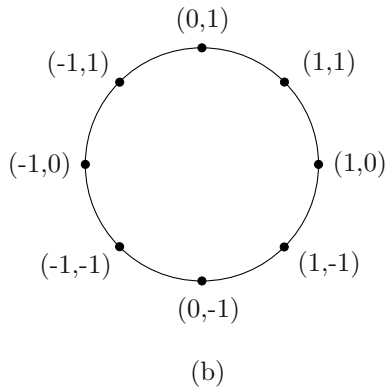
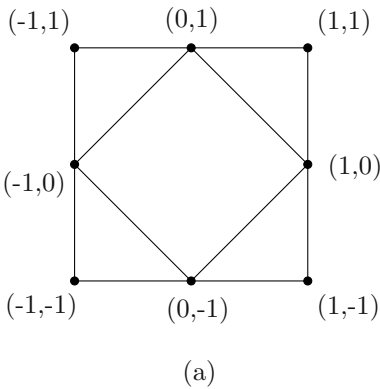


Fig. 3. Label diagram for 2-fans. (a) Transitions. (b) Circular diagram

If p_a and p_b have the same color, then $\mu(p, d) = \mu(p, d')$ and $\alpha(\phi) = \alpha(\phi')$. We assume that p_a and p_b have different colors. Then $|n_i - n'_i| \leq 1$ for $i = 1, 2$. There are eight possible values for $\alpha(p, d)$ and $\alpha(p, d')$ — the vertices of the diagram in Fig. 3 (a). It can be easily verified that the edges of the label diagram represent all possible transitions from $\alpha(p, d)$ to $\alpha(p, d')$.

We associate a circular diagram with the labels $\alpha()$ by placing them at the vertices of the regular 8-gon on the unit circle as shown in Fig. 3 (b). Let p be a point in the interior of a face $f \in \mathcal{A}$. Let $\pi(p)$ be the closed path of $\alpha(p, d)$ on the circular diagram, when d rotates 360° in clockwise order. We define the winding number $\omega(p)$ as the number of clockwise turns of the path $\pi(p)$.

Lemma 2. Let p be a point in the interior of a face $f \in \mathcal{A}$. Suppose that there is no equitable 2-fan with center p . Then the winding number $\omega(p)$ is odd.

Suppose that there is no equitable 2-fan with center p . The winding number is well defined since every transition is represented in the circular diagram as $\pm 1/8$ or $\pm 1/4$ of the one clockwise turn. We show that $\omega(p)$ is odd. Let $(p; d_1, d'_1)$ be a 2-fan. Let $\pi_1 = \alpha(p, d_1), \alpha(p, d_2), \dots, \alpha(p, d_l)$ be the path on the label diagram when the starting direction of a 2-fan rotates from d_1 to $d_l = d'_1$ in clockwise order. Note that $\alpha(p, d_l) = -\alpha(p, d_1)$. Thus, the path π_1 makes $t + 1/2, t \in \mathbb{Z}$ clockwise turns on the label diagram. Let π_2 be the path on the label diagram when the starting direction of a 2-fan rotates from d_l to d_1 in clockwise order. It has property that $\pi_2 = \alpha(p, d_l) = -\alpha(p, d_1), -\alpha(p, d_2), \dots, -\alpha(p, d_l) = \alpha(p, d_1)$. Thus, π_2 makes $t + 1/2, t \in \mathbb{Z}$ clockwise turns on the label diagram. The winding number $\omega(p)$ is $2t + 1$. The lemma follows.

Lemma 2 allows us to define the winding number of a face f , denoted by $\omega(f)$, assuming that an equitable 2-fan does not exist. Since the paths $\pi(p)$ are the same for all points p in the face f , we denote it by $\pi(f)$.

Lemma 3. Let e be an edge of a face $f \in \mathcal{A}$. Let f_1, f_2 be the faces adjacent to e . Then $\omega(f_1) = \omega(f_2)$.

The edge e is an arc on a great circle C passing through two points of P , say p_a and p_b . The points p_a and p_b partition the great circle C into two great arcs A_1 and A_2 such that one, say A_1 , is shorter than the other. If e is a part of A_1 , then the paths $\pi(f_1)$ and $\pi(f_2)$ are equal. Suppose that $e \subseteq A_2$.

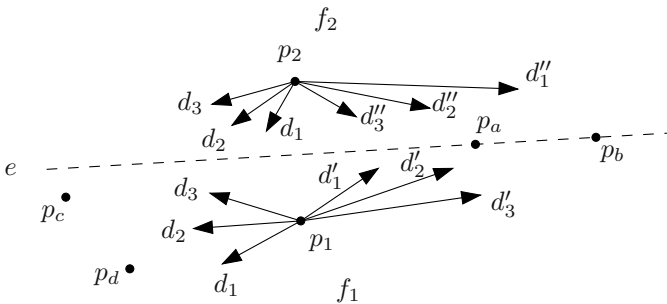


Fig. 4. Lemma 3

Let $p_i, i = 1, 2$ be a point in the face f_i . Let $\phi_j = (p_1; d_j, d'_j), j = 1, 2, 3$ be three consecutive 2-fans (ϕ_{j+1} is the CW-neighbor of ϕ_j) with center p_1 such that the lune $\sigma_1(p_1, d'_1, d'_2)$ contains p_a , see Fig. 4. Let p_c and p_d be two other points that change σ_1 of these three 2-fans. The points of P in the lunes σ_1 transform between 2-fans of the path $\pi(f_1)$ as follows:

$$\Sigma \cup \{p_c, p_d\} \rightarrow \Sigma \cup \{p_a, p_c\} \rightarrow \Sigma \cup \{p_a, p_b\}$$

where Σ is the set of points in P that stay in σ_1 . The corresponding transformations for the point p_2 are

$$\Sigma \cup \{p_c, p_d\} \rightarrow \Sigma \cup \{p_b, p_c\} \rightarrow \Sigma \cup \{p_a, p_b\},$$

see Fig. 4.

If p_a and p_b have the same color then the transformations for p_1 are the same as for p_2 and, thus, the paths $\pi(p_1)$ and $\pi(p_2)$ are equal. Suppose that the colors of p_a and p_b are different. Without loss of generality we assume that p_a is red and p_b is not red. Let $n_1 = \mu_1(p_1, d_1)$. It is easily verified that,

(i) if p_d is red, then $\mu_1(p_i, d_j) \in \{n_1 - 1, n_1\}$ for all $i = 1, 2$ and $j = 1, 2, 3$, and

(ii) if p_d is not red, then $\mu_1(p_i, d_j) \in \{n_1, n_1 + 1\}$ for all $i = 1, 2$ and $j = 1, 2, 3$.

Therefore, the numbers $\alpha_1(p_i, d_j), i = 1, 2, j = 1, 2, 3$ either all avoid -1 or all avoid +1. This implies that the paths $\alpha(p_1, d_1), \alpha(p_1, d_2), \alpha(p_1, d_3)$ and $\alpha(p_2, d_1), \alpha(p_2, d_2), \alpha(p_2, d_3)$ make the same (possibly fractional) number of turns.

Note that the path $\pi(p_1)$ may differ from $\pi(p_2)$ in two places since the arc A_1 can be swept by either of great half-circles of a 2-fan $(p; d, d')$. The case where $c(p, d')$ intersects A_1 is similar. The lemma follows.

The arrangement \mathcal{A} is defined using $\binom{2N}{2}$ great circles. Clearly, for each face f of \mathcal{A} , its symmetric image about the center of the sphere is a face of \mathcal{A} . We call it the *antipodal face* of f .

Theorem 3 (Antipodal Mapping). *If f_1 and f_2 are antipodal faces of \mathcal{A} then $\omega(f_2) = -\omega(f_1)$.*

Let p_1 be a point of the face f_1 . Then $p_2 = -p_1$ is the point of the face f_2 . Let $q_0, q_1, \dots, q_{2N-1}$ be the points of P in clockwise order from the point of view p_1 . Let $P_i, 0 \leq i < 2N$ be the set $\{q_i, q_{i+1}, \dots, q_{i+N}\}$ (we assume that the indices are modulo $2N$). Let π_i denote $(\delta(|P_i \cap R| - n), \delta(|P_i \cap B| - m))$. Then $\pi(p_1) = \pi_0, \pi_1, \dots, \pi_{2N-1}$. On the other hand $\pi(p_2) = \pi_{2N-1}, \pi_{2N-2}, \dots, \pi_0$ since the clockwise order of the points of P from the point of view p_2 is $q_{2N-1}, q_{2N-2}, \dots, q_0$. Therefore the path $\pi(p_2)$ makes $-\omega(p_1)$ number of turns. The theorem follows.

Lemma 3 and Theorem 3 imply Theorem 2. The continuous versions of Theorem 1 and 2 can be shown using standard techniques, see for example [3, 6, 19].

4 Algorithm for Finding 2-Fans

We show that an equitable 2-fan whose existence is guaranteed by Theorem 2 can be computed efficiently.

Theorem 4. *Let P be a set of $2N$ points on the sphere S . Then there is an algorithm that*

- (i) *computes the winding number $\omega(p)$ for each point $p \in P$ in $O(N \log^2 N)$ time,*
- (ii) *finds an equitable 2-fan in $O(N)$ time.*

By rotating the points on the sphere we can assume that the given great circle is $\{p \in S \mid p_z = 0\}$. Theorem 2 implies that the great half-circle $\gamma = \{(\cos \psi, \sin \psi, 0) \mid \pi/2 \leq \psi \leq 3\pi/2\}$ contains the center of an equitable 2-fan. Let $\gamma_0 = (0, 1, 0)$ and $\gamma_1 = (0, -1, 0)$ be the endpoints of γ .

Let p be a point on the sphere S . The winding number of p can be found in $O(N \log N)$ time by (i) sorting the points of P in clockwise order, and (ii) computing the path $\pi(p)$ on the label diagram (note that the algorithm can stop here if $\alpha() = 0$ is found), and (iii) computing $\omega(p)$.

There are $N(2N - 1)$ great circles in the arrangement \mathcal{A} since every two points of P lie on the unique great circle. The great circles of \mathcal{A} intersect γ in at most $N(2N - 1)$ points. These points partition γ into the arcs A_1, A_2, \dots, A_M where $M \leq N(2N - 1) + 1$. One can apply the binary search on the arcs. Let $A_i, i = \lfloor M/2 \rfloor$ be the median arc and let p be a point in A_i . Compute $\omega(p)$ and compare it with $\omega(\gamma_0)$ and $\omega(\gamma_1)$. One of the arcs γ_0p or $p\gamma_1$ (or both) satisfies the property that its endpoints have different winding numbers. We can proceed with this arc. By Lemma 3 the algorithm finds an equitable 2-fan.

We actually apply the binary search for two sub-arrangements of \mathcal{A} . Let \mathcal{A}_+ , resp. \mathcal{A}_- , be the arrangement of the great circles passing through pairs of points $P \cap S_+$, resp. $P \cap S_-$, where $S_+ = \{(x, y, z) \in S \mid z > 0\}$ is the upper hemisphere of S and $S_- = \{(x, y, z) \in S \mid z < 0\}$ is the lower hemisphere of S . The binary search used in the algorithm is a combination of two searches in \mathcal{A}_+ and \mathcal{A}_- , see details later. Our algorithm relies on the fact that the great circle $\gamma(p_a, p_b)$ generated by a point p_a in S_+ and a point p_b in S_- can be ignored since the path $\pi(p)$ does not change when p crosses $\gamma \cap \gamma(p_a, p_b)$. Therefore we can restrict ourself to the arrangements \mathcal{A}_+ and \mathcal{A}_- only. The question now is how to find the i -th arc on γ generated by \mathcal{A}_+ or \mathcal{A}_- . We show that this can be done using the slope selection.

The points on the upper hemisphere S_+ can be mapped to the plane $\Pi = \{(x, y, z) \mid z = 1\}$ using the gnomonic projection $\eta : S_+ \rightarrow \Pi$ defined as $\eta(x, y, z) = (x/z, y/z, 1)$. Let $p_a = (x_a, y_a, z_a)$ and $p_b = (x_b, y_b, z_b)$ be two points on the upper hemisphere and let $\gamma(p_a, p_b)$ denote the great circle containing p_a and p_b . The great circle $\gamma(p_a, p_b)$ is mapped to the line $l(p_a, p_b)$ passing through the points $\eta(p_a)$ and $\eta(p_b)$. We show that the sorted order of the lines $l(p_a, p_b)$ by slope is in the one-to-one correspondence with the order of the points $\gamma \cap \gamma(p_a, p_b)$ on the half-circle γ . The slope of the line $l(p_a, p_b)$ is

$$s = \frac{y_b/z_b - y_a/z_a}{x_b/z_b - x_a/z_a}.$$

Let $p_\gamma(x, y, 0)$ be the intersection point of the arc γ and the great circle $\gamma(p_a, p_b)$. Since the points $p_\gamma, 0, p_a$ and p_b are coplanar we have

$$\begin{vmatrix} 0 & 0 & 0 & 1 \\ x & y & 0 & 1 \\ x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \end{vmatrix} = 0.$$

Then $x(y_a z_b - y_b z_a) = y(x_a z_b - x_b z_a)$ and the slope is $s = y/x$. Note that $y/x = \tan \psi$ where ψ is the angle of $0p_\gamma$. The one-to-one correspondence between the slopes of the lines $l(p_a, p_b), p_a \in S_+, p_b \in S_-$ and the points $\gamma \cap \gamma(p_a, p_b)$ follows from the fact that $\psi \in (\pi/2, 3\pi/2)$ and $\tan(\cdot)$ is the monotone function in this interval.

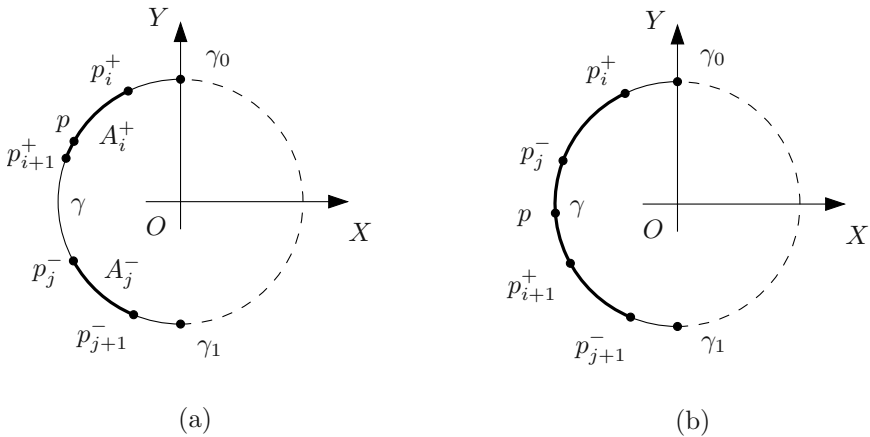


Fig. 5. The binary search step

Let $A_1^+, A_2^+, \dots, A_{m^+}^+$ be the arcs on γ into which γ is partitioned by the great circles of \mathcal{A}_+ . Similarly, let $A_1^-, A_2^-, \dots, A_{m^-}^-$ be the arcs generated by \mathcal{A}_- . We assume that the arcs A_i^+ (and the arcs A_j^-) are in counterclockwise order. Using the slope selection we can find the i -th arc A_i^+ . The slope selection can be done in $O(n \log n)$ time [8, 9] (a randomized algorithm can be found in [11, 18]).

In the binary search step, we have the sequence of arcs $A_{i_1}^+, A_{i_1+1}^+, \dots, A_{i_2}^+$ and the sequence of arcs $A_{j_1}^-, A_{j_1+1}^-, \dots, A_{j_2}^-$. At the beginning the indices are $i_1 = j_1 = 1, i_2^+ = m^+$ and $j_2^- = m^-$. Using the slope selection we find two arcs $A_i^+ = (p_i^+, p_{i+1}^+)$ and $A_j^- = (p_j^-, p_{j+1}^-)$ where $i = \lfloor (i_1 + i_2)/2 \rfloor$ and $j = \lfloor (j_1 + j_2)/2 \rfloor$. Notice that the arc A_i^+ may contain many arcs generated by \mathcal{A} . Therefore the points of A_i^+ may have different winding numbers.

Suppose that A_i^+ and A_j^- are disjoint. If the sequence of arcs of \mathcal{A}_+ is narrowed to just the arc A_i^+ , i.e. $i_1 = i_2$, then we make the decision for \mathcal{A}_- . In the example depicted in Fig. 5 (a), the indices of the arcs in \mathcal{A}_- should be less than j . If the sequence of arcs of \mathcal{A}_+ has more than one segment then we pick a point p in the arc A_i^+ infinitesimally close to p_{i+1}^+ and test it. If $\omega(p) = \omega(\gamma_0)$ then we prune the arcs in \mathcal{A}_+ with indices less than i . Otherwise we prune the arcs in \mathcal{A}_+ with indices greater than i and the arcs in \mathcal{A}_- with indices greater than j .

If the arcs A_i^+ and A_j^- intersect then we select any point p from their intersection, see Fig. 5 (b). Note that all the points of $A_i^+ \cap A_j^-$ have the same winding number since the path $\pi(p)$ does not change when p crosses $\gamma \cap \gamma(p_a, p_b)$ for any $p_a \in S_+, p_b \in S_-$. The binary search test for p prunes either the arcs of \mathcal{A}_+ or \mathcal{A}_- (or both).

The total number of the binary search tests is $O(\log N)$ since every time the arcs of \mathcal{A}_+ or \mathcal{A}_- are pruned. The theorem follows.

The problem of finding an equitable 2-fan on the sphere is more general than the one in the plane since we can map the plane $\Pi = \{(x, y, z) \mid z = 1\}$ to the sphere S using the inverse gnomonic projection η^{-1} which maps a plane point $(x, y, 1)$ to the sphere point (xz, yz, z) , where $z = 1/\sqrt{x^2 + y^2 + 1}$. As an immediate consequence we obtain the following result.

Theorem 5. *Let M be a set of $2N$ masses on the sphere. Then there exists an equitable 2-fan with center in M that can be computed in time $O(N \log^2 N)$ (i) and $O(N)$ (ii) if the masses are in general position.*

5 Conclusion

We proved that any three mass measures on the 2-sphere admit an equitable partition by a 2-fan whose center lies on a given great circle. Based on our proof we showed that an equitable 2-fan can be computed in $O(N \log^2 N)$ time for discrete masses. The complexity of the problem is an interesting open question: one can either improve the running time (a new proof may be needed for this) or/and find a non-trivial lower bound.

References

1. J. Akiyama, A. Kaneko, M. Kano, G. Nakamura, E. Rivera-Campo, S. Tokunaga, and J. Urrutia. Radial perfect partitions of convex sets in the plane. In *Proc. Japan Conf. Discrete Comput. Geom.*'98, volume 1763 of *Lecture Notes Comput. Sci.*, pp. 1–13. Springer-Verlag, 2000.
2. I. Bárány. Geometric and combinatorial applications of Borsuk's theorem. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pp. 235–249. Springer-Verlag, 1993.
3. I. Bárány and J. Matoušek. Simultaneous partitions of measures by k -fans. *Discrete Comput. Geom.*, 25(3):317–334, 2001.

4. I. Bárány and J. Matoušek. Equipartition of two measures by a 4-fan. *Discrete Comput. Geom.*, 27(3):293–301, 2002.
5. M. W. Bern and D. Eppstein. Multivariate regression depth. *Discrete Comput. Geom.*, 28(1):1–17, 2002.
6. S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink. Generalizing ham sandwich cuts to equitable subdivisions. *Discrete Comput. Geom.*, 24(4):605–622, 2000, <http://springerlink.metapress.com/openurl.asp?genre=article&issn=0179-5376&volume=24&issue=4&spage=605>.
7. P. Bose, E. D. Demaine, F. Hurtado, J. Iacono, S. Langerman, and P. Morin. Geodesic ham-sandwich cuts. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, 2004, <http://www-ma2.upc.es/~hurtado/ham.pdf>.
8. H. Brönnimann and B. Chazelle. Optimal slope selection via cuttings. *Comput. Geom. Theory Appl.*, 10(1):23–29, 1998.
9. R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.
10. M. Díaz and J. O’Rourke. Ham-sandwich sectioning of polygons. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pp. 282–286, 1990.
11. M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. *Internat. J. Comput. Geom. Appl.*, 2:1–27, 1992.
12. D. P. Dobkin and H. Edelsbrunner. Ham-sandwich theorems applied to intersection problems. In *Proc. 10th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, pp. 88–99, 1984.
13. H. Edelsbrunner and R. Waupotitsch. Computing a ham-sandwich cut in two dimensions. *J. Symbolic Comput.*, 2:171–178, 1986.
14. H. Ito, H. Uehara, and M. Yokoyama. 2-dimension ham sandwich theorem for partitioning into three convex pieces. In *Proc. Japan Conf. Discrete Comput. Geom. ’98*, volume 1763 of *Lecture Notes Comput. Sci.*, pp. 129–157. Springer-Verlag, 2000.
15. A. Kaneko and M. Kano. Balanced partitions of two sets of points in the plane. *Comput. Geom. Theory Appl.*, 13:253–261, 1999.
16. M. Kano. Personal communication.
17. C.-Y. Lo, J. Matoušek, and W. L. Steiger. Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, 11:433–452, 1994.
18. J. Matoušek. Randomized optimal algorithm for slope selection. *Inform. Process. Lett.*, 39:183–187, 1991.
19. J. Matoušek. *Using the Borsuk-Ulam Theorem*. Springer-Verlag, Heidelberg, 2003.
20. T. Sakai. Balanced convex partitions of measures in \mathbb{R}^2 . *Graphs and Combinatorics*, 18(1):169–192, 2002.
21. W. Steiger. Algorithms for ham sandwich cuts. In *Proc. 5th Canad. Conf. Comput. Geom.*, p. 48, 1993.
22. S. T. Vrećica and R. T. Živaljević. Conical equipartitions of mass distributions. *Discrete Comput. Geom.*, 25(3):335–350, 2001.
23. R. T. Živaljević. Topological methods. In J. E. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 11, pp. 209–224. CRC Press LLC, Boca Raton, FL, 1997.
24. R. T. Živaljević and S. T. Vrećica. An extension of the ham sandwich theorem. *Bull. London Math. Soc.*, 22:183–186, 1990.

Augmenting the Edge-Connectivity of a Spider Tree^{*}

Davide Bilò and Guido Proietti

Dipartimento di Informatica, Università di L'Aquila, Italy
{davide.bilo, proietti}@di.univaq.it

Abstract. Given an undirected, 2-edge-connected, and real weighted graph G , with n vertices and m edges, and given a spanning tree T of G , the 2-edge-connectivity augmentation problem with respect to G and T consists of finding a minimum-weight set of edges of G whose addition to T makes it 2-edge-connected. While the general problem is NP-hard, in this paper we prove that it becomes polynomial time solvable if T can be rooted in such a way that a prescribed topological condition with respect to G is satisfied. In such a case, we provide an $\mathcal{O}(n(m+h+\delta^3))$ time algorithm for solving the problem, where h and δ are the height and the maximum degree of T , respectively. A faster version of our algorithm can be used for 2-edge connecting a spider tree, that is a tree with at most one vertex of degree greater than two. This finds application in strengthening the reliability of optical networks.

Keywords: Graph Algorithms, Edge-Connectivity Augmentation, NP-hardness, Spider Tress, Network Survivability.

1 Introduction

Let $G = (V, E)$ be a connected, undirected graph, modelling the sites and the potential links of the layout of a communication network. Let us assume that a real weight is associated with each edge $e \in E$, representing the cost for activating the link. Clearly, the cheapest network that allows all the sites to communicate is a spanning tree of G . However, such a network is highly susceptible to failures, since it cannot survive even if just one link or site fails. Given current attention to network reliability, one desires to build more resilient networks.

A network having edge-connectivity $\lambda > 1$ continues to allow communication between functioning sites even after as many as $\lambda - 1$ links have failed. The extension of these definitions for the vertex-connectivity case is straightforward. In this paper, we give attention to the problem of strengthening the reliability of an already existing network, in the link failure case. More precisely, given a tree T , our purpose is to make it 2-edge-connected by simply adding a minimum-weight set of edges different from the ones in T . This optimization problem is NP-hard and 2-approximable for general graphs. But, as we will show in this

^{*} Work partially supported by the Research Project GRID.IT, funded by the Italian Ministry of Education, University and Research.

paper, it becomes polynomial if the given graph has a special structure with respect to the tree T .

A classical formulation of a λ -edge-connectivity augmentation problem (λECAP for short) is the following: Given a k -edge connected weighted graph G , given an h -edge connected spanning subgraph H of G , with $h < k$, and given an integer $h < \lambda \leq k$, find a minimum-weight set of edges of G , say $AUG_\lambda(H, G)$, whose addition to H increases its edge-connectivity to λ .

For $\lambda > 1$, the problem is NP-hard. A particular instance of the problem occurs when H has no edges. Such a problem, called λ -edge-connectivity augmentation problem, is APX-hard and not approximable within $\frac{68569}{68564} - \epsilon$, for any constant $\epsilon > 0$, even for $\lambda = 2$ [1]. Thus, a large amount of work has been done on designing approximation algorithms. When $h = 1$ and $\lambda = 2$, which is of interest for this paper, the best known approximation result for the general weighted case is 2. The first algorithm, by Frederickson and Jájá [5] was simplified later by Khuller and Thurimella [10] and recently, Galluccio and Proietti [8] have lowered the time complexity of the algorithm. For the unweighted case, algorithms with a best approximation ratio are known. More precisely, Nagamochi developed a $(1.875 + \epsilon)$ -approximation algorithm, for any constant $\epsilon > 0$ [11], then improved to $3/2$ by Even [4].

Besides, by adding additional constraints on the structure of H and/or G , researchers have characterized polynomial time solvable cases of the problem. First, Eswaran and Tarjan proved that $AUG_2(H, G)$ can be found in polynomial time if G is a complete unweighted graph [3]. This result has been extended to any desired edge-connectivity value by Watanabe and Nakamura [12], and faster algorithms can be found in [7]. For the weighted case, Galluccio and Proietti [8] have recently developed a fast polynomial time algorithm finding an optimal 2-edge-connectivity augmentation of a depth-first search tree of G .

In this paper, we move one step forward along this direction. More precisely, we show that if H is actually a spanning tree T of G , then the 2ECAP is polynomial time solvable if T can be rooted in such a way that the following property holds:

$$e = (u, v) \in T \implies u \text{ is an ancestor of } v \text{ in } T \text{ and } T(u) \text{ is } 2\text{-edge-connected.}$$

As we will show, this is not just a very special case. Indeed, a faster version of our algorithm can be used to 2-edge-connect at minimum cost a graph, that is a tree with at most one vertex having a degree greater than two. This is related to a reliability problem in optical communication networks. Furthermore, with opportune but realistic constraints on the structure of G , the above property captures several widespread network topologies, like caterpillar trees, bus networks, spider networks with appended spider trees, and many others, that will be extensively treated in the full version of this paper.

The paper is organized as follows: in Section 2 we give some basic definitions that will be used throughout the paper; in Section 3 we show our polynomial case of the 2ECAP, and we provide a corresponding solving algorithm; finally, in Section 4 we apply these results to augment a spider tree.

2 Basic Definitions

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and $E \subseteq \{(u, v) | u, v \in V \wedge u \neq v\}$ is the set of edges. If $(u, v) \in E$ we say that u is adjacent to v . G is said to be *simple* if there exists a real function $w : E \mapsto \mathbb{R}$ (to simplify the notation, we denote by $w(u, v)$ the weight of the edge (u, v)). Note that the definition of graph permits no loops, i.e., no edge joining a vertex to itself. In a *simple graph*, no loops are allowed, but more than one edge can join two vertices (*parallel edges*). If both loops and parallel edges are permitted, we have a *multigraph*. Given $v \in V$, the number of adjacent vertices to v with respect to the graph G , is the *degree* of v in G and it is denoted by $\delta_G(v)$.

Consider a set of edges E . The graph *induced* from E is the graph $G = (V, E)$, where each $v \in V$ is an endvertex of some $f \in E$. A graph $H = (V(H), E(H))$ is a *subgraph* of G if $V(H) \subseteq V$ and $E(H) \subseteq E$. If $V(H) = V$, then H is a *spanning subgraph* of G . The *weight* of H is defined as $w(H) = \sum_{e \in E(H)} w(e)$.

A *path* (or a *trail* for short) in G is a subgraph H of G with $V(H) = \{v_0, \dots, v_k | v_i \neq v_j \text{ for } i \neq j\}$ and $E(H) = \{(v_i, v_{i+1}) | 0 \leq i < k\}$, also denoted as $P(v_0, v_k)$. The *length* of a path is the number of its edges. A graph G is *connected* if, $\forall u, v \in V$, there exists a path $P(u, v)$ in G . The *connected components* of a graph G are the maximal (w.r.t. vertex inclusion) connected subgraphs of G .

A *spanning tree* T is a connected acyclic graph with a privileged vertex r called *root*. Let T denote a spanning tree of G rooted at r . Edges in T are called *tree edges*, while edges in $E \setminus E(T)$ are called *non-tree edges*. A non-tree edge (u, v) *crosses* all the tree edges along the (unique) path from u to v in T . By $C(e_1, e_2, \dots, e_k)$, with $k \geq 1$, we denote the set of non-tree edges such that each member covers the tree edges e_1, e_2, \dots, e_k .

Let $P(r, v)$ denote the unique path in T between r and $v \in V(T)$. Any vertex u in $P(r, v)$ is an *ancestor* of v in T , while v is a *child* of u . By $ANC(v)$ we denote the set of all ancestors of v . The (unique) vertex u in $P(r, v)$, $r \neq v$, preceding v is the *parent* of v and it is denoted by $p(v)$. Clearly, v is a *leaf* of u . A vertex with no child is a *leaf* of T . The *height* of T is the size of a maximum-size path out of the paths from r to each leaf of T . A connected subgraph of T is called *subtree* of T . Given $v \in V(T)$, by $T(v)$ we denote the subtree of T rooted at v containing all descendants of v . A graph whose connected components are trees, is a *forest*.

A graph G is said to be *2-edge-connected* if the removal of any edge from G leaves it connected. Given a spanning subgraph H of a 2-edge-connected graph G , solving the *2-Edge-Connected Augmentation Problem* (2ECAP) of H in G means to select a minimum-weight set of edges in $E \setminus E(H)$, denoted as $AUG_2(H, G)$, such that the spanning subgraph $H' = (V, E(H) \cup AUG_2(H, G))$ of G is 2-edge-connected. The extension of above definitions to multigraphs and pseudographs are straightforward.

In the rest of the paper, we will restrict our attention to the case in which H is connected. Notice that if H is connected, then, w.l.o.g., we can assume that H is a tree. Indeed, each 2-edge-connected component of H can be contracted into

a single This transforms the graph G into a multigraph \mathcal{G} and the graph H into a tree T whose edges are the bridges of H . It is then easy to see that finding $\text{AUG}_2(H, G)$ is equivalent to finding $\text{AUG}_2(T, \mathcal{G})$. Based on that, we will restrict ourselves to the problem of finding 2-edge-connectivity augmentation of trees over multigraphs. Trivially, we can also restrict ourselves to instances such that the weight of each non-tree edge is strictly positive, and in the case of parallel non-tree edges, we have to maintain only the cheapest one.

3 A Polynomial Case of the 2ECAP

Let T be a rooted spanning tree of $\mathcal{G} = (V, E)$. In this section, we examine the 2ECAP of T in \mathcal{G} , restricted to a special class of polynomial instances. To describe such a class, we need some new definitions.

Definition 1. $e = (p(v), v) \in E(T)$, dual graph G_e $V(G_e) = \{v_{e'} \mid e' = (v, u) \in E(T)\}$, $(v_{e'}, v_{e''}) \in E(G_e)$ $e' \quad e''$

Simply observing the definition above, we have that in [8] the authors optimally solve the 2ECAP over instances such that each tree edge has a dual graph made up by a forest of . . . (i.e., trees of height 1) and/or single vertices. Another interesting polynomial time class of instances arises when each dual graph is bipartite [2]. Our discussion restricted to still another class of instances, satisfying the following:

$$\forall e \in E(T), v_e \text{ has degree at most 1 in } G_e.$$

Before describing the optimal algorithm solving the problem for the class of instances satisfying Property 1, we introduce some new notations. For fixed $v \in V \setminus \{r\}$, by e_0 we denote the tree edge $(p(v), v)$. Moreover, by $v_1, v_2, \dots, v_{\delta_T(v)-1}$ we denote the children of v . We denote the tree edge (v, v_j) , $1 \leq j < \delta_T(v)$, by e_j . W.l.o.g., we can assume that in the dual graph, the vertex adjacent to the representative of e_0 is the representative of e_1 .

3.1 Graph Topology

Our first step in designing an optimal polynomial time algorithm is to formalize some new properties related to Property 1. This can be done by analyzing the topological structure of \mathcal{G} with respect to T .

Lemma 1. . . . \mathcal{G}, T $v \in V, \forall f_1 = (u_1, w_1), f_2 = (u_2, w_2) \in C(e_0, e_1)$ $u_1, u_2 \in V(T(v_1))$ $u_1 \in \text{ANC}(u_2) \vee u_2 \in \text{ANC}(u_1)$

Suppose the lemma is not true, that is $u_1 \notin \text{ANC}(u_2) \wedge u_2 \notin \text{ANC}(u_1)$. As $u_1, u_2 \in V(T(v_1))$, we have that the u for u_1 and u_2 in T is a vertex of $T(v_1)$. Since $\hat{e} = (p(u), u) \in E(T(v))$, then $f_1, f_2 \in C(\hat{e})$.

Let s_1 (resp., s_2) be the child of u in $P(u, u_1)$ (resp., $P(u, u_2)$). Let $\hat{e}_1 = (u, s_1)$ and $\hat{e}_2 = (u, s_2)$. As $s_1 \neq s_2$, it follows that $\hat{e}_1 \neq \hat{e}_2$. Consider the dual graph $G_{\hat{e}}$ and let $\hat{u}, \hat{u}_1, \hat{u}_2$ be the vertices in the dual graph corresponding to $\hat{e}, \hat{e}_1, \hat{e}_2$, respectively. We have that $(\hat{u}, \hat{u}_1) \in E(G_{\hat{e}})$, since $f_1 \in C(\hat{e}, \hat{e}_1)$. Moreover, $(\hat{u}, \hat{u}_2) \in E(G_{\hat{e}})$ since $f_2 \in C(\hat{e}, \hat{e}_2)$. Hence, such a dual graph does not satisfy Property 1. From this contradiction the claim follows. \square

From Lemma 1, the next two corollaries follow:

Corollary 1. $\mathcal{G}, T, v \in V, \forall (u_1, w_1), (u_2, w_2) \in C(e_j, e_k), 1 \leq j < k < \delta_T(v), u_1, u_2 \in T(v_j), T(v_k) (u_1 \in \text{ANC}(u_2) \vee u_2 \in \text{ANC}(u_1)) \wedge (w_1 \in \text{ANC}(w_2) \vee w_2 \in \text{ANC}(w_1))$ \square

Corollary 2. $\mathcal{G}, T, v \in V, \forall (u_1, w_1) \in C(e_j, e_k) \forall (u_2, w_2) \in C(e_k, e_i), 1 \leq j < k < i < \delta_T(v), u_2, w_1 \in T(v_k) w_1 \in \text{ANC}(u_2) \vee u_2 \in \text{ANC}(w_1)$ \square

Lemma 1 and Corollaries 1 and 2 characterize the topological structure of \mathcal{G} with respect to T , when Property 1 holds. Basically, they point out that, given any subtree $T(v)$ of T , the endvertices of all non-tree edges having only one endvertex in $T(v)$ are such that these endvertices lie along the same (unique) path $P(v, u)$, where u is a leaf of $T(v)$. We will call this path the \dots .

3.2 Basic Idea

By simply considering our class of instances, it is easy to see that every weighted multigraph defined over a star satisfies Property 1, so is a member of our class. Let us start by solving such simple instance. Consider the weighted pseudograph \mathcal{P} obtained from a copy of \mathcal{G} by contracting each tree edge e_i into a node v_i (see Figure 1). Clearly, edges parallel to tree edges become loops in \mathcal{P} .

Now, it is not hard to see that an augmentation for T in \mathcal{G} is represented in \mathcal{P} by an edge cover \mathcal{C} , i.e., a set of edges that spans all vertices of \mathcal{P} . Moreover, an edge cover \mathcal{C} for \mathcal{P} represents an augmentation for T in \mathcal{G} . Since the transformation described above preserves the weights, we have that:

Lemma 2. \dots minimum-cost edge cover (\dots) $\mathcal{P} \dots$ T, \mathcal{G}, \dots \square

3.3 The Algorithm

We aim to use the idea described in the previous subsection to realize an optimal algorithm solving a general instance.

High-Level Description. Our algorithm works in a bottom-up manner with respect to the height of the subtrees. It starts by finding a cover for the subtrees of T of height 1, then it proceeds with those of height 2, and so on, until it finds a cover for T itself (we will see later that we must do more).

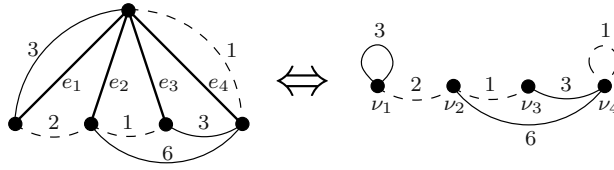


Fig. 1. An example of the transformation from the 2ECAP of a star versus an MCEC problem over a pseudograph and viceversa. Bold edges represent tree-edges, while dashed edges represent an optimal solution for both problems

Let h be the height of T . For $t = 1, 2, \dots, h$, consider the set $V_t = \{v \in V \mid T(v) \text{ has height } t\}$. Now fix a vertex $v \in V_t$. Since we have already found a solution for each $T(v_j)$, $1 \leq j < \delta_T(v)$, we can transform $T(v)$ into a rooted star $\mathcal{T}(v)$ by contracting each $T(v_j)$ into a leaf of $\mathcal{T}(v)$, while the set of all the other vertices are contracted into the root of $\mathcal{T}(v)$. If we apply this transformation to \mathcal{G} , all non-tree edges covering e_j , for some $1 \leq j < \delta_T(v)$, are kept.

By applying the same idea of Section 3.2, we find an optimal cover for $e_1, e_2, \dots, e_{\delta_T(v)-1}$. Now it is easy to see that this cover together with a cover for each $T(v_j)$, $1 \leq j < \delta_T(v)$, is a cover for $T(v)$. Moreover every cover for $T(v)$ contains edges forming a cover for $\mathcal{T}(v)$. However, this transformation does not guarantee optimality (see Figure 2). To avoid this problem we simply need to adjust the weights of non-tree edges before the contraction phase.

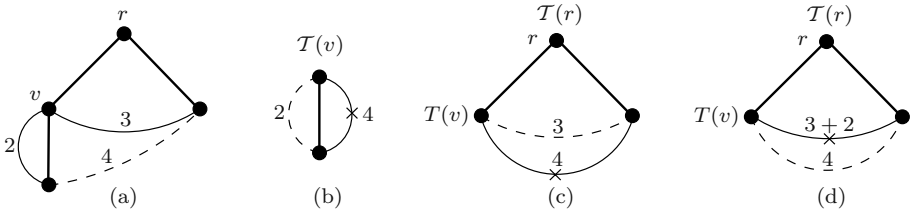


Fig. 2. A simple execution of the algorithm with and without tuning the weight. For each example, bold edges are tree edges, while dashed edges represent optimal solution. In (a), the input instance, where the optimum has weight 4; in (b), the star representing $T(v)$; in (c), the star representing $T(r)$ without adjusting the weights: notice that the returned cover for T has weight 5; in (d), the star representing $T(r)$ after tuning the weights: note that the returned cover for T has weight 4 (equal to the optimum in (a))

Tuning the Weights. Let $T(v, i)$, $0 \leq i \leq t$, be the forest induced by the set of edges of $T(v)$ deprived of the first i edges of the v -path, starting from v (see Figure 3). Now, fix a node $v \in V_t$, and consider $f \in C(e_j)$, with $1 \leq j < \delta_T(v)$. The non-tree edge f covers a subpath of the v_j -path containing v_j . Assume that the size of this path is i . The tree edges in $T(v_j)$ not covered by f induce the forest $T(v_j, i)$. We say that $T(v_j, i) \dots$ to f in $T(v_j)$. If $f \in C(e_j, e_k)$, then the forest \dots to f in $T(v)$ is the union of the forests appended to f in $T(v_j)$ and $T(v_k)$, respectively.

A natural way for updating the weight of f , is charging f with the weight of the solution computed for the appended forest w.r.t the considered tree (see Figure 2). So, for any $v \in V_t$, we need to find a cover for $T(v, i)$, with $0 \leq i \leq t$, denoted as $SOL(v, i)$. Observe that $T(v) = T(v, 0)$. To save memory space, $SOL(v, i)$ will not be computed explicitly, but we associate with $T(v, i)$ a set $X(v, i) \subseteq SOL(v, i)$, which will allow to rebuild $SOL(v, i)$ by simply merging it with the solutions computed at previous steps.

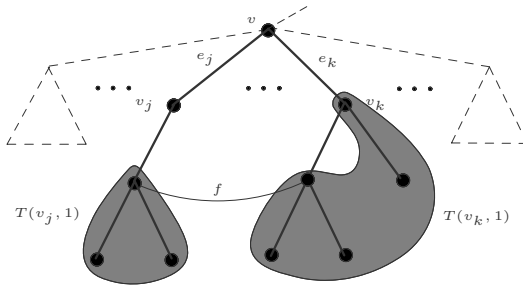


Fig. 3. An example of appended forest. Bold and dashed edges represent tree-edges while dashed triangle represent subtrees. Left and right shaded area are the forests $T(v_j, 1)$, $T(v_k, 1)$ appended to f in $T(v_j)$, $T(v_k)$, respectively. The graph induced by $T(v_j, 1) \cup T(v_k, 1)$ is the forest appended to f in $T(v)$

To simplify notations, if $f \in C(e_j)$, then by $SOL(f, e_j)$ we denote the computed solution for the forest appended to f in $T(v_j)$ (if this forest is the empty graph, then we assume that the weight of its cover is 0). If $f \in C(e_j, e_k)$, then by $SOL(f) = SOL(f, e_j) \cup SOL(f, e_k)$ we denote the computed solution for the forest appended to f in $T(v)$. The updating rule is formally defined as follows:

Definition 2. $v \in V_t, \dots, f \in C(e_j), \dots, 1 \leq j < \delta_T(v)$

$$\phi(e_j, f) := w(f) + w(SOL(f, e_j)). \tag{1}$$

selected edge $C(e_j) \dots \dots \dots \epsilon(e_j) \dots \dots \dots$

$$\phi(e_j, \epsilon(e_j)) = \min_{f \in C(e_j)} \{ \phi(e_j, f) \}. \tag{2}$$

Definition 3. $v \in V_t, \dots, f \in C(e_j, e_k), \dots, 1 \leq j < k < \delta_T(v)$

$$\varphi(f) := w(f) + w(SOL(f)). \tag{3}$$

selected edge $C(e_j, e_k) \dots \dots \dots \epsilon(e_j, e_k) \dots \dots \dots$

$$\varphi(\epsilon(e_j, e_k)) = \min_{f \in C(e_j, e_k)} \{ \varphi(f) \}. \tag{4}$$

Note that for each non-tree edge $f \in C(e_j, e_k) = C(e_j) \cap C(e_k)$, we compute $\phi(e_j, f), \phi(e_k, f), \varphi(f)$. This is a useful trick that allows us to see f as a cover for either of e_j, e_k , or both edges.

Building the Pseudographs. Now we can apply the same transformation described in Section 3.2. Remember that we have to compute a cover for $T(v, i)$, with $0 \leq i \leq t$. Moreover, observe that if $i > 0$, then $T(v, i)$ does not contain the edge e_1 . In this case, a cover for $T(v_1, i - 1)$ can be used.

So, for each $0 \leq i \leq t$, we build a weighted pseudograph $P_i = (V_i, E_i)$, such that $V_i = \{\nu_1, \nu_2, \dots, \nu_{\delta_T(v)-1}\}$ and $E_i = \{(\nu_j, \nu_j) | 1 \leq j < \delta_T(v)\} \cup \{(\nu_j, \nu_k) | C(e_j, e_k) \neq \emptyset, j \neq k\}$. With each edge in E_i , we associate a real weight in the following way:

$$\omega_i(f) := \begin{cases} \varphi(\epsilon(e_j, e_k)) & \text{if } f = (\nu_j, \nu_k) \in E_i, j \neq k; \\ w(\text{SOL}(v_1, i - 1)) & \text{if } f = (\nu_1, \nu_1) \text{ and } i > 0; \\ \phi(e_j, \epsilon(e_j)) & \text{if } f = (\nu_j, \nu_j) \in E_i. \end{cases}$$

Let \mathcal{C}_i be the solution of the minimum-cost edge cover problem (MCECP) defined over the weighted pseudograph P_i . The set $X(v, i)$ is given by the selected edges represented by the edges in \mathcal{C}_i . More formally, $X(v, i) = A(v, i) \cup B(v, i)$ is given by:

$$A(v, i) = \bigcup_{\substack{(\nu_j, \nu_k) \in \mathcal{C}_i \\ j \neq k}} \{\epsilon(e_j, e_k)\}; \quad B(v, i) = \begin{cases} \bigcup_{(\nu_j, \nu_j) \in \mathcal{C}_i} \{\epsilon(e_j)\} & \text{if } i = 0; \\ \bigcup_{\substack{(\nu_j, \nu_j) \in \mathcal{C}_i \\ j \neq 1}} \{\epsilon(e_j)\} & \text{if } i \neq 0. \end{cases}$$

The set $\text{SOL}(v, i)$ can be obtained from $X(v, i)$ in the following way:

$$\text{SOL}(v, i) = X(v, i) \cup \bigcup_{f \in A(v, i)} \text{SOL}(f) \cup \bigcup_{(\nu_j, \nu_j) \in \mathcal{C}_i} \text{SOL}(\epsilon(e_j), e_j).$$

It is easy to see that $\omega(\mathcal{C}_i) = w(\text{SOL}(v, i))$.

3.4 Correctness of the Algorithm

At the end of the execution, the algorithm clearly returns a covering for T , so the set $\text{SOL}(r, 0)$ is a 2-edge-connectivity augmentation of T . Then, we must prove that $\text{SOL}(r, 0)$ is optimal.

Theorem 1. $\text{SOL}(r, 0)$ is an optimal covering for T .

The proof proceeds by induction on the height of T . For $t = 1, 2, \dots, h$, we will show that $\forall v \in V_t, \text{SOL}(v, i)$ is an optimal covering for $T(v, i), 0 \leq i \leq t$.

($t = 1$) $T(v, i)$ is a tree of height at most 1. Moreover, $\forall f \in C(e_j), 1 \leq j < \delta_T(v)$, we have that $\phi(e_j, f) = w(f)$, and $\forall f \in C(e_j, e_k), 1 \leq j < k < \delta_T(v)$, we have $\varphi(f) = w(f)$. Moreover, if $i > 0$, then $w(\text{SOL}(v_1, i - 1)) = 0$ since $T(v_1)$ is a tree made by a single vertex. Then, the claim follows from the correctness of the minimum-cost edge cover algorithm over pseudographs.

($t > 1$) Suppose that $\forall v \in V_s$, with $1 \leq s < t \leq h$, $\text{SOL}(v, i)$, $0 \leq i \leq s$, is an optimal solution for $T(v, i)$. This explains why at edges (ν_1, ν_1) in \mathcal{P}_j , $1 \leq j \leq t$, we assign the weight of the computed cover for $T(\nu_1, i - 1)$ (by induction this weight is optimal!). Now consider a node $v \in V_t$. We want to prove that, it suffices to choose only one non-tree edge from each $C(e_j, e_k)$, $1 \leq j < k < \delta_T(v)$. For the sake of contradiction, suppose this assertion is not true. Then, $\exists f_1 = (u_1, w_1), f_2 = (u_2, w_2) \in C(e_j, e_k)$ such that:

$$\begin{aligned} \varphi(f_1), \varphi(f_2) &> w(f_1) + w(f_2) + \min\{w(\text{SOL}(f_1, e_j)), w(\text{SOL}(f_2, e_j))\} + \\ &+ \min\{w(\text{SOL}(f_1, e_k)), w(\text{SOL}(f_2, e_k))\}. \end{aligned} \tag{5}$$

From Corollary 1, we can consider the following cases:

1. $u_1 \in \text{ANC}(u_2) \wedge w_1 \in \text{ANC}(w_2)$ (the case $u_2 \in \text{ANC}(u_1) \wedge w_2 \in \text{ANC}(w_1)$ is symmetric).

The forest appended to f_2 in $T(v)$ is a subgraph of the forest appended to f_1 in $T(v)$. From (5) and by induction it follows that

$$w(f_2) + w(\text{SOL}(f_2)) = \varphi(f_2) > w(f_1) + w(f_2) + w(\text{SOL}(f_2))$$

from which we derive that $w(f_1) < 0$. As $w(f_1) > 0$, we get a contradiction.

2. $u_1 \in \text{ANC}(u_2) \wedge w_2 \in \text{ANC}(w_1)$ (the case $u_2 \in \text{ANC}(u_1) \wedge w_1 \in \text{ANC}(w_2)$ is symmetric).

The forest appended to f_2 in $T(v_j)$ is a subgraph of that appended to f_1 in $T(v_j)$, while the forest appended to f_1 in $T(v_k)$ is a subgraph of that appended to f_2 in $T(v_k)$. Since by induction $w(\text{SOL}(f_2, e_k)) \leq w(f_1) + w(\text{SOL}(f_1, e_k))$, from (5) it follows that

$$\begin{aligned} \varphi(f_2) &> w(f_1) + w(f_2) + w(\text{SOL}(f_2, e_j)) + w(\text{SOL}(f_1, e_k)) \\ &\geq w(f_2) + w(\text{SOL}(f_2)) = \varphi(f_2). \end{aligned}$$

We have then obtained a contradiction.

Analogously we can prove that it suffices to choose only one edge from each $C(e_j)$, with $1 \leq j < \delta_T(v)$. Now, since every edge cover for \mathcal{P}_i can be transformed into a cover for $T(v, i)$ with the same overall weight (and viceversa), the claim follows. □

3.5 Analysis of the Algorithm

The following lemma helps us to improve the performance of the algorithm.

Lemma 3. *Let h be the height of T . Let $v \in V_t$, $0 \leq t \leq h$, P_0, P_1, \dots, P_t be the path of T from v to the root P_0 . Let C_i be the set of non-tree edges of \mathcal{P}_i , $1 \leq i \leq t$. Let C_0, C_1, \dots, C_t be the set of non-tree edges of T such that $C_i \cap C_j = \emptyset$, $i \neq j$.*

Some important properties of the built pseudographs are the following:

- (i) $V_i = V_j$ and $E_i = E_j$, with $0 \leq i < j \leq t$;
- (ii) $\forall (\nu_j, \nu_k) \in E_i$, with $0 \leq i \leq t$, $\omega_i(\nu_j, \nu_j), \omega_i(\nu_k, \nu_k) \leq \omega_i(\nu_j, \nu_k)$;
- (iii) $\omega_j(\nu_1, \nu_1) \leq \omega_i(\nu_1, \nu_1)$, with $0 \leq i < j \leq t$ (it follows from the correctness of the algorithm);
- (iv) For every edge $f \neq (\nu_1, \nu_1)$, $\omega_i(f) = \omega_j(f)$, with $0 \leq i < j \leq t$, (in this case we will omit the index of ω).

Let P' be the weighted pseudograph obtained from a copy of P_0 by removing vertex ν_1 and all its incident edges. Let C' be an optimal edge cover for P' . We can consider the following cases:

1. $(\nu_1, \nu_1) \in C_0$. Since $C_0 = C' \cup \{(\nu_1, \nu_1)\}$, then from property (iii) and (iv) it trivially follows that C_0 is an MCEC for $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_t$, and the claim is true.
2. $(\nu_1, \nu_i) \in C_0$, for some $1 < i < \delta_T(v)$. Note that if $(\nu_1, \nu_i), (\nu_1, \nu_j) \in C_0$ ($i \neq j$), then from property (ii) we have that $C_0 \cup (\nu_j, \nu_j)$ minus (ν_1, ν_j) is an optimal edge cover for \mathcal{P}_0 . Let C^* an optimal edge cover for \mathcal{P}_0 with only one edge (say (ν_1, ν_i)) incident to ν_1 . Let P'' be the weighted pseudograph obtained from a copy of P_0 by deleting vertices ν_1, ν_i and all their incident edges. Let C'' be an optimal edge cover for P'' . Now, we have that

$$\omega(C^*) = \omega(\nu_1, \nu_i) + \omega(C'') \leq \omega_0(\nu_1, \nu_1) + \omega(C').$$

To decrease the cost of the edge cover, we must necessarily have that $\omega(\nu_1, \nu_i) + \omega(C'') > \omega_j(\nu_1, \nu_1) + \omega(C')$, for some $0 < j \leq t$. Since $\omega_j(\nu_1, \nu_1) \leq \omega_0(\nu_1, \nu_1)$ implies $\omega_j(\nu_1, \nu_1) + s = \omega_0(\nu_1, \nu_1)$, with $s \geq 0$, and since $\omega(C'') \leq \omega(C')$ implies $\omega(C'') + s' = \omega(C')$, with $s' \geq 0$, it follows that

$$\begin{aligned} \omega_j(\nu_1, \nu_1) + \omega(C') + s' &= \omega_0(\nu_1, \nu_1) - s + \omega(C') + s' \\ &< \omega(\nu_1, \nu_i) + \omega(C'') + s' = \omega(\nu_1, \nu_i) + \omega(C'), \end{aligned}$$

from which it follows that

$$\omega(\nu_1, \nu_i) - \omega_0(\nu_1, \nu_1) > s' - s. \tag{6}$$

Thus, by computing C_0, C^* and C' , we can compute $\omega(C'') = \omega(C^*) - \omega(\nu_1, \nu_j)$ and so we can compute s' . Since from (6) we can compute s , we can find the minimum $0 < j^* \leq t$ such that $\omega_{j^*}(\nu_1, \nu_1) + s \leq \omega_0(\nu_1, \nu_1)$. Therefore, we have that C_0 is an MCEC for $\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_{j^*-1}$, while $C_{j^*} = \{(\nu_1, \nu_1)\} \cup C'$ is an MCEC for $\mathcal{P}_{j^*}, \mathcal{P}_{j^*+1}, \dots, \mathcal{P}_t$. The claim follows. \square

Now we can prove the following:

Theorem 2. $\text{AUG}_2(T, \mathcal{G}) = O(n(m + h + \delta^3))$

n T T
 m \mathcal{G} δ T
 h \mathcal{G} δ T

We can use the algorithm described in Section 3.3 to compute $\text{AUG}_2(T, \mathcal{G})$. The correctness of the algorithm derives from Theorem 1.

To establish the time complexity of the algorithm, first note that an MCECP over a weighted pseudograph with n vertices and m edges can be solved optimally (see [13]) in $\mathcal{O}(n(m + n \lg n))$ time and space [6]. Now observe that, $\forall v \in V$, the algorithm selects the edges $\epsilon(e_j)$, for every $1 \leq j < \delta_T(v)$, and the edges $\epsilon(e_j, e_k)$, for every $1 \leq j < k < \delta_T(v)$. This task can be accomplished in $\mathcal{O}(m)$ time and space. Let h' be the height of $T(v)$. The algorithm creates at most h' weighted pseudographs with at most $\delta_T(v) - 1 = \mathcal{O}(\delta)$ vertices. Since these pseudographs have no multiple edges, it follows that the number of edges is $\mathcal{O}(\delta^2)$. For each pseudograph, the algorithm computes a minimum-cost edge cover. Using Lemma 3, this task can be accomplished in $\mathcal{O}(h' + \delta^3) = \mathcal{O}(h + \delta^3)$. Therefore, the total running time is $\mathcal{O}(n(m + h + \delta^3))$. \square

4 Augmenting a Spider Tree

The results of the previous sections have an interesting application for solving the 2ECAP of a spider tree, i.e., a tree with at most one vertex, called δ , with degree greater than 2.

The problem of finding a spanning spider of a given graph G , which is NP-hard, arises from a problem in optical networks [9]. The wavelength division multiplexing technology of optical communication, supports the propagation of multiple laser beams through a single optical fiber, as long as each beam has a different wavelength. A new technology in optical networks allows a switch to replicate the optical signal by splitting light, thus allowing to extend the light-path concept as to incorporate optical multicasting capability. Multicast is the ability to transmit information from a single source node to multiple destination nodes, and many bandwidth-intensive applications, such as worldwide web browsing, video conferencing, video on demand services, etc., require multicasting for efficiency purposes.

The switches correspond then to nodes of degree greater than two (called δ). However, typical optical networks will have a limited number of these sophisticate switches, and one has to settle them in such a way that all possible multicasts can be performed. Thus, we are led to the problem of finding spanning trees with as few branch vertices as possible. A spider tree is exactly a spanning tree with a minimum number of branch vertices.

But, as observed in the introduction, a spider tree structure is not resilient to edge failures. Hence, it makes sense to consider the problem of increasing its edge-connectivity to 2, by adding a minimum-cost (with respect to some given weight function) set of edges, chosen among a set of possible candidates. This corresponds to solve a 2ECAP for a spider tree T rooted at the center. We can prove the following:

Theorem 3. Let $\mathcal{G} = (V, E)$ be a weighted graph with n vertices and m edges. Let T be a spider tree rooted at the center of \mathcal{G} with δ branch vertices and $m' = \mathcal{O}(\delta^2)$ edges.

$$\text{AUG}_2(T, \mathcal{G}) \leq \mathcal{O}(m \cdot \alpha(n, n) + \delta(m' + \delta \log \delta)) \cdot T(r, \alpha(\cdot, \cdot))$$

Trivially, T satisfies Property 1. Thus, we can optimally solve the problem in polynomial time (see Theorem 2).

About the time complexity, note that every possible appended forest in \mathcal{G} , is made up of subtrees of T , so it suffices to compute only the value $T(v, 0) = T(v)$, for each $v \in V$. Let us consider the root r . Subtrees $T(v_1), T(v_2), \dots, T(v_b)$, are simply disjoint paths. For path of n vertices, one can find an optimal 2-edge-connectivity augmentation in $\mathcal{O}(m \cdot \alpha(m, n))$ time and space [8], where m is the number of additional edges. Let $n_1, n_2, \dots, n_\delta$ denote the number of vertices of $T(v_1), T(v_2), \dots, T(v_\delta)$, respectively. Let $m_1, m_2, \dots, m_\delta$ denote the number of edges having an endvertex in $T(v_1), T(v_2), \dots, T(v_\delta)$, respectively. For each tree $T(v_i)$, with $1 \leq i \leq \delta$, we find an optimal cover using the algorithm described in [8]. Since $\sum_{i=1}^b n_i = n - 1$ and $\sum_{i=1}^b m_i \leq 2m$, it follows that $\sum_{i=1}^b \mathcal{O}(m_i \cdot \alpha(m_i, n_i)) = \mathcal{O}(m \cdot \alpha(n, n))$. Finally, we consider the vertex r , and we use our algorithm. First, we update the non-tree edges and select a subset of them (let m' be the cardinality of such set), and then we find an MCEC over a pseudograph with δ vertices and m' edges in $\mathcal{O}(\delta(m' + \delta \log \delta))$ time [6]. \square

References

1. H.-J. Böchenhauer, D. Bongartz, J. Hromkovič, R. Klasing, G. Proietti, S. Seibert and W. Unger, On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality, *Theoretical Computer Science*, to appear. A preliminary version was presented at *22nd Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'02)*, Vol. 2556 of Lecture Notes in Computer Science, Springer-Verlag, 59–70.
2. M. Conforti, A. Galluccio, and G. Proietti, Augmentation problems and network matrices, *30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'04)*. Proceedings will appear on Lecture Notes in Computer Science, Springer-Verlag.
3. K.P. Eswaran and R.E. Tarjan, Augmentation problems, *SIAM Journal on Computing*, **5**(4) (1976), 653–665.
4. G. Even, J. Feldman, G. Kortsarz, and Z. Nutov, A 3/2-approximation algorithm for augmenting the edge-connectivity of a graph from 1 to 2 using a subset of a given edge set, *4th Int. Workshop on Approximation Algorithms For Combinatorial Optimization (APPROX 2001)*, Vol. 2129 of Lecture Notes in Computer Science, Springer-Verlag, 90–101.
5. G.N. Frederickson and J. JàJà, Approximation algorithms for several graph augmentation problems, *SIAM Journal on Computing*, **10**(2) (1981), 270–283.
6. H.N. Gabow, Data structures for weighted matching and nearest common ancestors with linking, *1st Symp. on Discrete Algorithms (SODA 1990)*, 434–443.
7. H.N. Gabow, Application of a poset representation to edge-connectivity and graph rigidity, *32nd IEEE Symp. on Found. of Computer Science (FOCS'91)*, 812–821.

8. A. Galluccio and G. Proietti, Polynomial time algorithms for 2-edge-connectivity augmentation problems, *Algorithmica*, **36**(4) (2003), 361–374.
9. L. Gargano, P. Hell, L. Stacho, and U. Vaccaro, Spanning trees with bounded number of branch vertices, *29th Int. Coll. on Automata, Languages and Programming (ICALP '02)*, Vol. 2380 of Lecture Notes in Computer Science, Springer-Verlag, 355–365.
10. S. Khuller and R. Thurimella, Approximation algorithms for graph augmentation, *Journal of Algorithms*, **14**(2) (1993), 214–225.
11. H. Nagamochi, An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree, *Discrete Applied Mathematics*, **126**(1) (2003) 83–113.
12. A. Nakamura and T. Watanabe, Edge-connectivity augmentation problems, *Journal of Computer and System Science*, **35**(1) (1987), 96–144.
13. C.H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, New Jersey, 1982.

On Nash Equilibria for Multicast Transmissions in Ad-Hoc Wireless Networks

Vittorio Bilò, Michele Flammini, Giovanna Melideo, and Luca Moscardelli

Dipartimento di Informatica,
Università di L'Aquila,
Via Vetoio, Coppito 67100 L'Aquila
{bilo, flammini, melideo, moscardelli}@di.univaq.it

Abstract. We study a multicast game in ad-hoc wireless networks in which a source sends the same message or service to a set of receiving stations via multi-hop communications and the overall transmission cost is divided among the receivers according to given cost sharing methods. Receivers enjoy a benefit equal to the difference between the utility they get from the transmission and the shared cost they are asked to pay. Assuming a selfish and rational behavior, each user is willing to receive the transmission if and only if his shared cost does not exceed his utility. Moreover, given the strategies of the other users, he wants to select a strategy of minimum shared cost. A Nash equilibrium is a solution in which no user can increase his benefit by seceding in favor of a different strategy. We consider the following reasonable cost sharing methods: the overall transmission cost is equally shared among all the receivers (egalitarian), the cost of each intermediate station is divided among its downstream receivers equally (semi-egalitarian) or proportionally to the transmission powers they require to reach their next-hop stations (proportional), and finally each downstream user at an intermediate station equally shares only his required next-hop power among all the receivers asking at least such a level of power (Shapley value). We prove that, while the first three cost sharing methods in general do not admit a Nash equilibrium, the Shapley value yields games always converging to a Nash equilibrium. Moreover, we provide matching upper and lower bounds for the price of anarchy of the Shapley game with respect to two different global cost functions, that is the overall cost of the power assignment, that coincides with the sum of all the shared costs, and the maximum shared cost paid by the receivers. Finally, in both cases we show that finding the best Nash equilibrium is computationally intractable, that is NP-hard.

1 Introduction

Wireless networks have received significant attention during recent years. In particular, they emerged due to their potential applications in emergency disaster relief, battlefield, etc [19, 24]. Unlike traditional wired networks or cellular networks, they do not require the installation of any wired backbone infrastructure.

Any station of the network has a fixed position and is equipped with an omnidirectional antenna which is responsible for sending and receiving signals. Communications occur by assigning to each station a transmission power. In order to minimize the overall energy consumption, communications between stations may require multi-hop transmissions, that is relaying on intermediate stations. Thus, a naturally arising issue is that of supporting communication patterns that are typical of traditional networks with minimum cost (see [3] for a survey). We are mainly interested in one-to-many communications or multicasting, in which the message or service of a given source station s must be forwarded to users residing at a subset of receiving stations.

Multicasting data to a large population is likely to incur significant costs that is natural to be shared in some fashion among receivers. We assume that each receiving station t has a certain utility u_t representing the benefit he gets from receiving the service from the source. Moreover, receivers act selfishly and thus they are only interested in maximizing their own benefit and will agree to pay for the transmission if and only if they are charged a cost not exceeding the benefit they get from it.

There is a vast literature on implementation paradigms in which cost-sharing mechanisms are designed to achieve the socially desirable outcomes in spite of users selfishness. Namely, a cost sharing mechanism determines which users receive the service and at what price. Many of these approaches assume that utilities u_t , because they are considered private properties of receivers, and that, as receivers are selfish, they may misreport utilities in order to maximize their benefit. In such a case the network can discourage such deception by using strategy-proof cost sharing mechanisms, that is mechanisms in which the optimal (dominant) strategy for each receiver is to truthfully reveal his utility no matter how the other receivers behave, and strategy-proof mechanisms, where this holds for coalitions as well.

The multicast cost sharing problem has been largely investigated in standard networks in [8, 9, 10, 13], where the cost shared among the receivers is the total cost of the links of a multicast subtree connecting the source to all the receivers and the cost of a network link is equally shared among its down-streaming users. Recently, the problem has been also considered in wireless networks [1, 18], where the cost shared among the receivers is the overall power consumption.

In this paper we focus on the strategy-proof approach in a context where receivers' utilities u_t are considered properties of the wireless network and, as a consequence, are a priori well-known. Unlike strategy-proofness above, where a mechanism leads to an equilibrium in which no receiver can improve its utility by deviating no matter how the other receivers behave, Nash equilibria assume that selfish receivers reach a self-consistent equilibrium, called the Nash equilibrium, in which no one can improve its own benefit, given the behavior of the others. Indeed, each user is interested in receiving from the source with a minimum shared cost, given the choices of the other users. For such reasons, the evolution of the interactions among the receivers' choices gives rise to a multi-player game in which each player (station) t has a set of strategies equal to the

set of all the directed (s, t) -paths in the network and a payoff function equal to the difference between his utility and the shared cost according to its chosen path and the ones of the other receivers. Called a *game*, the set of the chosen (s, t) -paths for every receiving station t , a set of feasible solutions \mathcal{N} for a multi-player game \mathcal{G} is the set of its Nash equilibria [17], that is the set of the path systems such that no player has an incentive to secede in favor of a different solution.

The main new algorithmic issues coming from this model are the following:

1. Proving the non-emptiness of \mathcal{N} ; ¹
2. Proving the convergence to a Nash equilibrium from any initial configuration of the players' strategies and estimating the convergence time;
3. Finding Nash equilibria having particular properties (for instance, the one minimizing the global cost);
4. Measuring the *price of anarchy* [14], that is the ratio between the worst Nash equilibrium and the *price of cooperation*, that is the optimal solution that could be achieved if all players cooperated.

Several games [6, 7, 11, 16, 20, 23] have been shown to possess pure Nash equilibria or to converge to a pure Nash equilibrium independently from their starting state. An interesting work estimating the convergence time to Nash equilibria is [5] and in [4] finding Nash equilibria having particular properties has been shown to be NP-complete. Finally, considerable research effort has been also devoted to analyzing the price of anarchy in different settings [2, 15, 21].

In this paper we consider the games induced by four reasonable cost sharing methods \mathcal{M} which distribute the cost of the transmission powers of the intermediate stations among the receivers as follows:

1. in an *equal* way, that is by equally distributing the overall cost among receivers;
2. in a *down-streaming* way, that is by equally distributing the transmission cost of each station t among its down-streaming receivers;
3. in a *proportional* way, that is by distributing the transmission cost of each station t among its down-streaming receivers proportionally to the transmission powers they require to reach their next-hop stations;
4. by applying the definition of the *Shapley value* [22], depending on the different levels of power necessary for exchanging messages between t and the next-hop stations (see Section 2); more precisely, the cost up to a given transmission power is equally shared among all the receivers requiring *at least* such a level of power for transmitting to their next-hop stations.

We prove that, while the first three cost sharing methods in general do not admit a Nash equilibrium, the Shapley value yields games always converging to a Nash equilibrium starting from any initial configuration. Then, we provide

¹ Indeed, Nash proved that a randomized equilibrium always exists, while we are interested in pure Nash equilibria.

matching upper and lower bounds for the price of anarchy of the Shapley game with respect to two different global cost functions, that is the overall cost of the power assignment, which coincides with the sum of all the shared costs, and the maximum shared cost paid by the receivers. Unfortunately, in both cases the price of anarchy is the worst possible one, that is equal to the number of receivers. Finally, in both cases we show that finding the best Nash equilibria is computationally intractable, that is NP-hard.

The paper is organized as follows. In the next section we present the basic definitions and notation. In Section 3 we prove the results on the non existence of Nash equilibria for the egalitarian, semi-egalitarian and proportional cost sharing methods and the convergence of the Shapley value one. In Section 4 we bound the price of anarchy of the Shapley game according to the two above mentioned global functions and in Section 5 we show the intractability of determining the best Nash equilibrium. Finally, in Section 6 we give some conclusive remarks and discuss some open questions.

2 Definitions and Notation

A wireless network is usually modelled as a complete graph $G(S, c)$ in which $S = \{1, \dots, n\}$ is a set of radio stations and $c : S \times S \mapsto \mathbb{R}^+$ is a symmetric function associating to each pair of stations t and t' in S a value $c(t, t')$, that is the power necessary for exchanging messages between t and t' . Clearly, $c(t, t) = 0$ for every station $t \in S$. Since each user potentially interested in receiving the transmission resides in some station t , from now on we identify each user with a corresponding receiving station.

Given a distinguished source station $s \in S$, for any station $t \in S$, let \mathcal{P}_t be the set of all the possible directed (s, t) -paths in G and p_t be any (s, t) -path in \mathcal{P}_t .

A power assignment $\omega : S \mapsto \mathbb{R}^+$ to the stations induces a directed weighted graph $G_\omega = (S, A)$, called *power graph*, such that an arc (t, t') belongs to A if and only if the transmission power of t is at least equal to the transmission cost from t to t' , i.e., $\omega(t) \geq c(t, t')$. The *total cost* of a power assignment ω is the overall power consumption yielded by ω , i.e.,

$$cost(\omega) = \sum_{t \in S} \omega(t).$$

Given a set \mathcal{P} of directed paths in G from the source s , also called a *path system*, let $\omega_{\mathcal{P}}$ be the power assignment of minimal cost such that $G_{\omega_{\mathcal{P}}}$ contains all the paths in \mathcal{P} . Namely, denoted as $A(\mathcal{P})$ the set of the arcs belonging to at least a path in \mathcal{P} , $\omega_{\mathcal{P}}(t) = \max_{(t, t') \in A(\mathcal{P})} c(t, t')$ for every $t \in S$. For the sake of simplicity we denote $\omega_{\mathcal{P}}$ as ω_p when $\mathcal{P} = \{p\}$.

Once a path system \mathcal{P} has been fixed for a subset of receivers R , with a little abuse of notation we often consider each $p_t \in \mathcal{P}$ as the set of the stations met along p_t , thus writing $t' \in p_t$ if t' is in the directed path of t . We define as $l(\mathcal{P}, t') = |\{t \in R \mid t' \in p_t\}|$ the number of downstream receivers of station t'

and as $l(\mathcal{P}, t', \varpi) = |\{t \in R \mid t' \in p_t \wedge \omega_{p_t}(t') \geq \varpi\}|$ the number of such receivers requiring a transmission of power greater or equal to ϖ at t' .

Let $P(t) = \{\varpi \in \mathbb{R}^+ \mid \exists t' \in S : c(t, t') = \varpi\}$ be the set of the possible power assignments for station t , $P_i(t)$ be the i -th smallest element of $P(t)$ (that is the i -th one in increasing order), and $L_i(t) = P_i(t) - P_{i-1}(t)$ (with $P_0(t) = 0$) for $2 \leq i \leq |P(t)|$, be the marginal power assignment needed for station t to increase its transmission power from $P_{i-1}(t)$ to $P_i(t)$.

A *cost sharing method* \mathcal{M} is a function that, given any subset of receivers R with an associated path system \mathcal{P} , defines the amount of power consumption or cost $\mathcal{M}(\mathcal{P}, t)$ attributed to each receiver $t \in R$.

Definition 1. A *cost sharing method* \mathcal{M} is a function that, given any subset of receivers R with an associated path system \mathcal{P} , defines the amount of power consumption or cost $\mathcal{M}(\mathcal{P}, t)$ attributed to each receiver $t \in R$.
 $\mathcal{M}(\mathcal{P}, t) = 0 \quad t \notin R$
 $\sum_{t \in R} \mathcal{M}(\mathcal{P}, t) = cost(\omega_{\mathcal{P}})$

We will consider four fundamental cost sharing methods $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ and \mathcal{M}_4 defined as follows:

- The cost charged to each receiver $t \in R$ according to \mathcal{M}_1 is $\mathcal{M}_1(\mathcal{P}, t) = \frac{cost(\omega_{\mathcal{P}})}{|R|}$, that is the overall cost of the power assignment is equally shared among all the receivers.
- The cost charged to each receiver $t \in R$ according to \mathcal{M}_2 is $\mathcal{M}_2(\mathcal{P}, t) = \sum_{t' \in p_t} \frac{\omega_{\mathcal{P}}(t')}{l(\mathcal{P}, t')}$, that is the cost of a given $t' \in S$ is equally shared among all the receivers using t' as intermediate station.
- The cost charged to each receiver $t \in R$ according to \mathcal{M}_3 is $\mathcal{M}_3(\mathcal{P}, t) = \sum_{t' \in p_t} \omega_{\mathcal{P}}(t') \frac{\omega_{p_t}(t')}{\sum_{t'' : t' \in p_{t''}} \omega_{p_{t''}}(t')}$, that is the cost of a given $t' \in S$ is shared among all the receivers using t' as intermediate station proportionally to power they require.
- The cost charged to each receiver $t \in R$ according to \mathcal{M}_4 is $\mathcal{M}_4(\mathcal{P}, t) = \sum_{t' \in p_t} \sum_{i: P_i(t') \leq \omega_{p_t}(t')} \frac{L_i(t')}{l(\mathcal{P}, t', P_i(t'))}$, that is the marginal power $L_i(t')$ is equally shared among all the receivers using t' as intermediate station and having next-hop transmission cost at least equal to $P_i(t')$.

Given a utility profile $u = \langle u_1, \dots, u_{|S|-1} \rangle$, the *benefit* of any station $t \in S \setminus \{s\}$ is $b_t(u, \mathcal{P}, \mathcal{M}) = u_t - \mathcal{M}(\mathcal{P}, t)$ if $t \in R$, otherwise (t does not receive service) $\mathcal{M}(\mathcal{P}, t) = 0$ and its benefit is zero.

The multicast transmission game $\mathcal{G} = (G, u, \mathcal{M})$ is a $(|S| - 1)$ -player game, where each player $t \in S \setminus \{s\}$ has:

- a set of strategies $\mathcal{P}_t \cup \{\perp\}$ where \perp denotes the “empty” path, meaning that t is not interested in receiving the transmission (i.e., $p_t = \perp$ iff $t \notin R$), and
- a payoff function $b_t(u, \mathcal{P}, \mathcal{M}) = u_t - \mathcal{M}(\mathcal{P}, t)$ with $\mathcal{P} = \bigcup_{t \in S \setminus \{s\}} (\mathcal{P}_t \cup \{\perp\})$.

Such a game requires the selection of a (s, t) -path $p_t \in \mathcal{P}_t \cup \{\perp\}$ for each station $t \in S \setminus \{s\}$ and, given the set of receivers $R = \{t \mid t \in S \setminus \{s\} \wedge p_t \neq \perp\}$

and the path system $\mathcal{P} = \{p_t | t \in R\}$, the adoption of the corresponding power assignment $\omega_{\mathcal{P}}$.

A Nash equilibrium for \mathcal{G} is a path system \mathcal{P} such that, for a given set of receivers R :

- (a) $\forall t \in R, b_t(u, \mathcal{P}, \mathcal{M}) \geq 0,$
- (b) $\forall t \in R$ and path $p'_t \in \mathcal{P}_t$ inducing a new path system $\mathcal{P}' = \mathcal{P} \setminus \{p_t\} \cup \{p'_t\}$, it holds $\mathcal{M}(\mathcal{P}, t) \leq \mathcal{M}(\mathcal{P}', t),$
- (c) $\forall t \notin R$ and $p_t \in \mathcal{P}_t, \mathcal{M}(\mathcal{P} \cup \{p_t\}, t) \geq u_t.$

Consider the directed graph having a node for any possible path system \mathcal{P} and an arc $(\mathcal{P}, \mathcal{P}')$ whenever \mathcal{P} and \mathcal{P}' differ only for the choice of one receiver t and $b_t(u, \mathcal{P}, \mathcal{M}) > b_t(u, \mathcal{P}', \mathcal{M})$. If such a graph is acyclic, then every possible evolution of the game, also called *Nash dynamics*, converges, since a Nash equilibrium is always reached after a finite number of steps from any initial power assignment.

Denoting with \mathcal{N} the set of all the possible Nash equilibria for the game \mathcal{G} , the price of anarchy $\rho(\mathcal{G}, g)$ with respect to a given global function g is defined as the worst case ratio among the Nash versus optimal performance, that is $\rho(\mathcal{G}, g) = \max_{\mathcal{P} \in \mathcal{N}} \frac{g(\mathcal{P})}{g(\mathcal{P}^*)}$ when g has to be minimized and $\rho(\mathcal{G}, g) = \min_{\mathcal{P} \in \mathcal{N}} \frac{g(\mathcal{P})}{g(\mathcal{P}^*)}$ in case of maximization, where \mathcal{P}^* is an optimal path system with respect to g .

3 Existence of Nash Equilibria

In this section we first show that there are instances for which the three games $\mathcal{G} = (G, u, \mathcal{M}_1), \mathcal{G} = (G, u, \mathcal{M}_2)$ and $\mathcal{G} = (G, u, \mathcal{M}_3)$ do not admit any Nash equilibrium. Due to lack of space we omit the proof.

Theorem 1. *For any instance $\mathcal{G} = (G, u, \mathcal{M}_1), \mathcal{G} = (G, u, \mathcal{M}_2)$ and $\mathcal{G} = (G, u, \mathcal{M}_3)$ there does not exist any Nash equilibrium.*

We now prove that the multicast transmissions game yielded by the cost sharing method \mathcal{M}_4 defined in the previous section is always guaranteed to converge to a Nash equilibrium starting from any initial power assignment for the stations in S .

Theorem 2. *For any instance $\mathcal{G} = (G, u, \mathcal{M}_4)$ there exists a Nash equilibrium \mathcal{P}^* such that $u = \langle u_1, \dots, u_{|S|-1} \rangle$ and $G(S, c)$ is satisfied.*

We prove the claim by showing that the Nash dynamics is acyclic. Assume in fact by contradiction that there exists a cycle, that is a sequence of path systems $\langle \mathcal{P}^i \rangle$ with a corresponding sequence of power assignments $\langle \omega_{\mathcal{P}^i} \rangle$, for which there are two indexes j and k with $j < k$ such that $\mathcal{P}^j = \mathcal{P}^k$ and thus $\omega_{\mathcal{P}^j} = \omega_{\mathcal{P}^k}$.

According to the cost sharing method

$$\mathcal{M}_4(\mathcal{P}, t) = \sum_{t' \in \mathcal{P}_t} \sum_{q: P_q(t') \leq \omega_{\mathcal{P}_t}(t')} \frac{L_q(t')}{l(\mathcal{P}, t', P_q(t'))},$$

the choice performed at the generic step i by user t defines an inequality $in(i)$ which can be one of the following three possible types:

- (1) $u_t > \sum_{t' \in p_t^{i+1}} \sum_{q: P_q(t') \leq \omega_{p_t^{i+1}}^i(t')} \frac{L_q(t')}{l(\mathcal{P}, t', P_q(t'))}$
- (2) $\sum_{t' \in p_t^i} \sum_{q: P_q(t') \leq \omega_{p_t^i}^i(t')} \frac{L_q(t')}{l(\mathcal{P}, t', P_q(t'))} > \sum_{t' \in p_t^{i+1}} \sum_{q: P_q(t') \leq \omega_{p_t^{i+1}}^i(t')} \frac{L_q(t')}{l(\mathcal{P}, t', P_q(t'))}$
- (3) $\sum_{t' \in p_t^i} \sum_{q: P_q(t') \leq \omega_{p_t^i}^i(t')} \frac{L_q(t')}{l(\mathcal{P}, t', P_q(t'))} > u_t$.

The first inequality models the case in which user t , who was not receiving the transmission, detects a path $p_t^{i+1} \in \mathcal{P}_t$ yielding a new path system $\mathcal{P}^{i+1} = \mathcal{P}^i \cup \{p_t^{i+1}\}$ such that $\mathcal{M}_4(\mathcal{P}^{i+1}, t) < u_t$. The second inequality models the case in which user t decides to change his transmission path $p_t^i \in \mathcal{P}^i$ in favor of a cheaper one $p_t^{i+1} \in \mathcal{P}_t$, thus yielding a new path system $\mathcal{P}^{i+1} = \mathcal{P}^i \setminus \{p_t^i\} \cup \{p_t^{i+1}\}$ such that $\mathcal{M}_4(\mathcal{P}^{i+1}, t) < \mathcal{M}_4(\mathcal{P}^i, t)$. Finally, the third inequality models the case in which user t decides to secede from the solution since the cost of his transmission path $p_t^i \in \mathcal{P}^i$ has risen up to exceed his utility u_t , thus yielding a new path system $\mathcal{P}^{i+1} = \mathcal{P}^i \setminus \{p_t^i\}$ such that $\mathcal{M}_4(\mathcal{P}^{i+1}, t) = 0$.

The $k - j$ path choices performed from the path system \mathcal{P}^j to the path system \mathcal{P}^k thus define a consistent system of $k - j$ inequalities. The effects of a generic choice of user t at step i on the quantities $l(\mathcal{P}^{i+1}, t', \varpi)$ may be of the following types:

- if $t' \in p_t^i \cap p_t^{i+1}$ then $l(\mathcal{P}^{i+1}, t', \varpi) = l(\mathcal{P}^i, t', \varpi) - 1$ for any $\varpi \in P(t')$ such that $\omega_{p_t^{i+1}}^i(t') < \varpi \leq \omega_{p_t^i}^i(t')$ and $l(\mathcal{P}^{i+1}, t', \varpi) = l(\mathcal{P}^i, t', \varpi) + 1$ for any $\varpi \in P(t')$ such that $\omega_{p_t^i}^i(t') < \varpi \leq \omega_{p_t^{i+1}}^i(t')$;
- if $t' \in p_t^i \setminus p_t^{i+1}$ then $l(\mathcal{P}^{i+1}, t', \varpi) = l(\mathcal{P}^i, t', \varpi) - 1$ for any $\varpi \in P(t')$ such that $\varpi \leq \omega_{p_t^i}^i(t')$;
- if $t' \in p_t^{i+1} \setminus p_t^i$ then $l(\mathcal{P}^{i+1}, t', \varpi) = l(\mathcal{P}^i, t', \varpi) + 1$ for any $\varpi \in P(t')$ such that $\varpi \leq \omega_{p_t^{i+1}}^i(t')$.

Since $\mathcal{P}^j = \mathcal{P}^k$, we must have that for any $t \in S$ and $\varpi \in P(t)$, it holds $l(\mathcal{P}^j, t, \varpi) = l(\mathcal{P}^k, t, \varpi)$. If we plot the curve representing the evolution of the quantities $l(\mathcal{P}^i, t, \varpi)$ for any $\varpi \in P(t)$ for $j \leq i \leq k$ and draw an up-going (down-going) arrow at step i when $l(\mathcal{P}^i, t, \varpi)$ increases (decreases) at step i as shown in Figure 1, we can easily see that for each up-going arrow from the value x to the value $x + 1$ there is always a down-going one from the value $x + 1$ to the value x and viceversa. An up-going arrow at step i from the value x to the value $x + 1$ means also that the marginal transmitting power $L_q(t)$ appears in the righthand side of the summary of $in(i)$ with denominator equal to $l(\mathcal{P}^i, t, P_q(t)) = x + 1$, while a down-going arrow at step i from the value $x + 1$ to the value x means also that the marginal transmitting power $L_q(t)$ appears in the lefthand side of the summary of $in(i)$ with denominator equal to $l(\mathcal{P}^i, t, P_q(t)) = x + 1$. Finally, the absence of arrows at step i ($l(\mathcal{P}^i, t, P_q(t))$ stays the same) means that either t appears in both the lefthand and the righthand side of $in(i)$ with the same denominator or t does not appear in $in(i)$. This proves that for any station t and $\varpi \in P(t)$, the number of marginal power assignments $L_p(t)$ with a denominator

equal to x appearing in the lefthand side of all the inequalities defining the system is exactly equal to the number of marginal power assignments $L_p(t)$ with denominator equal to x appearing in the righthand side of all the inequalities defining the system. Moreover, $\mathcal{P}^j = \mathcal{P}^k$ implies that for any user t the same number of inequalities of type 1 and 3 must occur in the system, thus the same argument applies also to the values u_t . Hence, by summing up all the inequalities $in(i)$ for $j \leq i \leq k$ we obtain a strict inequality whose lefthand and the righthand sides are equal which is clearly unsatisfiable: a contradiction. \square

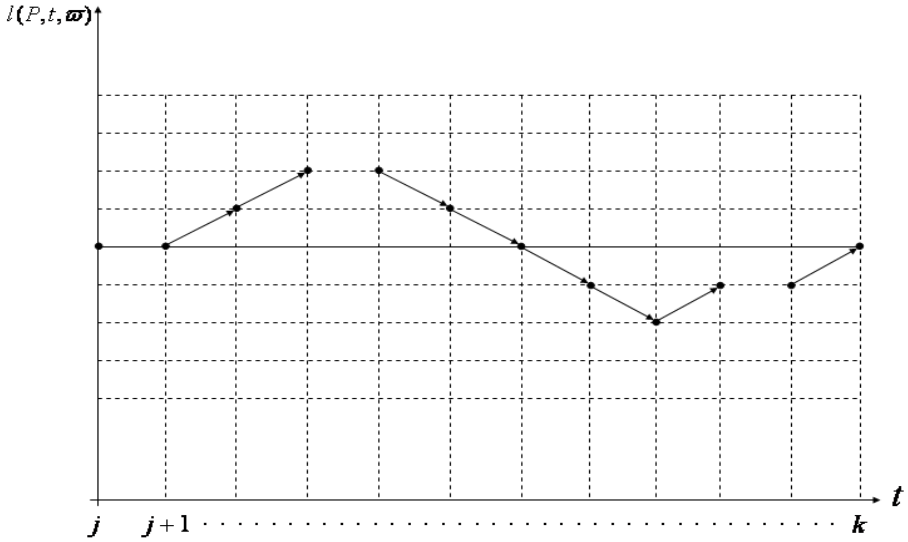


Fig. 1. Evolution of the quantities $l(\mathcal{P}^i, t, \varpi)$ in a cycle

Since our results show that the only converging game is yielded by the cost sharing method \mathcal{M}_4 , in the sequel of the paper we will restrict our attention only on the multicast transmission game $\mathcal{G} = (G, u, \mathcal{M}_4)$.

The following theorem characterizes the structure of any path system at Nash equilibrium for the game $\mathcal{G} = (G, u, \mathcal{M}_4)$.

Theorem 3. $\dots \dots \dots G(S, c) \dots \dots \dots u = \langle u_1, \dots, u_{|S|-1} \rangle, \dots \dots \dots \mathcal{P} \dots \dots \dots \mathcal{G} = (G, u, \mathcal{M}_4) \dots \dots \dots$

4 Bounding the Price of Anarchy

In this section we provide an upper and a lower bound on the price of anarchy for the game $\mathcal{G} = (G, u, \mathcal{M}_4)$.

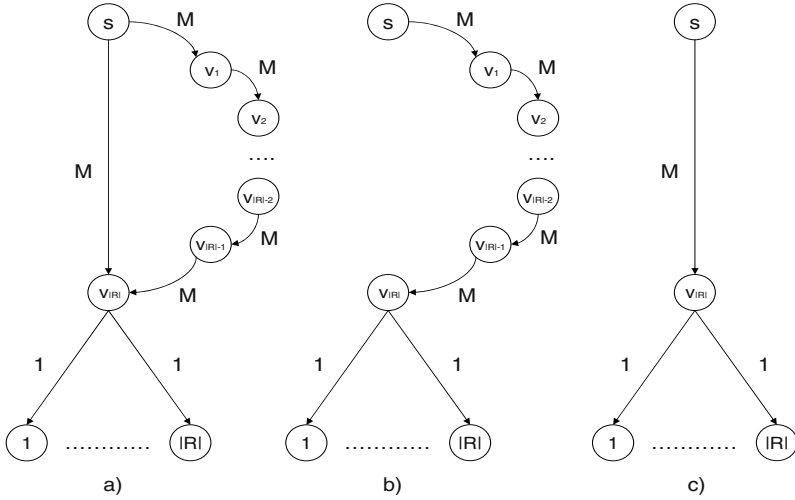


Fig. 2. a) The wireless network S ; b) The worst Nash equilibrium; c) The social optimum

In order to be consistent with the definition of price of anarchy, we restrict to global functions never assigning a null cost to a feasible solution. Moreover, we compare costs of solutions for fixed sets of receivers so as to obtain more realistic estimations.

Thus we focus on instances of the game in which the set R is part of the input, i.e. the receivers are interested in receiving the transmission at any cost, that is $u_t = \infty$ for any $t \in R$ and $u_t = 0$ for any $t \in S \setminus R$. The game $\mathcal{G} = (G, R, \mathcal{M}_4)$ is defined accordingly.

We define the following two global functions $g_{sum}(\mathcal{P}) = \sum_{t \in R} \mathcal{M}_4(\mathcal{P}, t) = cost(\omega_{\mathcal{P}})$ and $g_{max}(\mathcal{P}) = \max_{t \in R} \mathcal{M}_4(\mathcal{P}, t)$.

Theorem 4. $\rho(\mathcal{G}, g) \leq \frac{1}{|R|} \cdot g$ for $g \in \{g_{sum}, g_{max}\}$

Consider a path system at Nash equilibrium \mathcal{P} . For each $t \in R$, let $s^*(t)$ be the cost of a shortest (s, t) -path in G . Clearly, as \mathcal{P} is an equilibrium, it holds $\mathcal{M}_4(\mathcal{P}, t) \leq s^*(t)$. Since the total cost of a power assignment is exactly the sum of the costs charged to each receiver, we obtain $cost(\omega_{\mathcal{P}}) = \sum_{t \in R} \mathcal{M}_4(\mathcal{P}, t) \leq \sum_{t \in R} s^*(t) \leq |R| \cdot s^*(\bar{t})$, where $\bar{t} = \arg \max_{t \in R} s^*(t)$ is the receiver at maximum distance from s . Since it holds that $s^*(\bar{t}) \leq g_{sum}(\mathcal{P}^*)$, where \mathcal{P}^* is an optimum path system, we obtain the claim for $g = g_{sum}$. The claim for $g = g_{max}$ follows directly by considering that for any $t \in R$, it holds $s^*(t) \leq |R| \cdot g_{max}(\mathcal{P}^*)$. \square

Even though the above result is quite negative, in the following theorem we show a matching lower bound.

Theorem 5. $\rho(\mathcal{G}, g) = |R|$ for $g \in \{g_{sum}, g_{max}\}$ and $\mathcal{G} = (G, R, \mathcal{M}_4)$.

Consider the graph depicted in Figure 2a where $R = \{1, \dots, |R|\}$. It can be easily checked that the path system \mathcal{P} in Figure 2b is a Nash equilibrium of cost $g_{sum}(\mathcal{P}) = |R|M + 1$ and $g_{max}(\mathcal{P}) = M + \frac{1}{|R|}$, while the optimal solution \mathcal{P}^* for multicasting to R depicted in Figure 2c costs exactly $g_{sum}(\mathcal{P}^*) = M + 1$ and $g_{max}(\mathcal{P}^*) = \frac{M+1}{|R|}$. Letting M going to infinity, we obtain the claim. \square

5 Computing the Best Nash Equilibrium

In this section we prove that it is computationally hard to determine the best Nash equilibrium for the game $\mathcal{G} = (G, R, \mathcal{M}_4)$.

Theorem 6. $g \in \{g_{sum}, g_{max}\}$ and $\mathcal{G} = (G, R, \mathcal{M}_4)$ is NP-hard to compute $P = NP$.

We prove the claim by exploiting a reduction from the Exact 3-Cover problem [12]. In this problem we are given a universe $X = \{x_1, \dots, x_{3n}\}$ of $3n$ elements and a collection $\mathcal{C} = \{C_1, \dots, C_m\}$ of m subsets of X such that $|C_i| = 3$ for any $1 \leq i \leq m$ and $\bigcup_{i=1}^m C_i = X$. The objective is to find a collection of subsets $\mathcal{F} \subseteq \mathcal{C}$ such that $\mathcal{F} = \{F_1, \dots, F_n\}$ and $\bigcup_{i=1}^n F_i = X$.

Given an instance (X, \mathcal{C}) of Exact 3-Cover, we construct an instance of the multicast transmission game $\mathcal{G} = (G, R, \mathcal{M}_4)$ in the following way. $S = \{s, C_1, \dots, C_m, x_1, \dots, x_{3n}\}$, the set of edges connecting two stations at distance less than infinity is $E(S) = E_1 \cup E_2$, where $E_1 = \{(s, C_i), 1 \leq i \leq m\}$ and $E_2 = \{(C_i, x), (C_i, y), (C_i, z), \forall C_i = \{x, y, z\} \in \mathcal{C}\}$. Finally, we set $c(e) = 1$, for any $e \in E(S)$ and $R = \{x_i | 1 \leq i \leq 3n\}$.

Since in no Nash equilibrium a receiver can be reached by using two consecutive edges belonging to E_2 , every Nash equilibrium uses at least n edges belonging to E_1 and exactly $3n$ edges belonging to E_2 . Moreover, an exact 3-cover induces (and is induced by) a Nash equilibrium using exactly n edges belonging to E_1 . Thus, an equilibrium \mathcal{P}^* of minimum global cost is such that $g_{sum}(\mathcal{P}^*) = 1 + n$ and $g_{max}(\mathcal{P}^*) = \frac{n+1}{3n}$ if and only if there exists an exact 3-cover for X . \square

6 Conclusions

We have considered multicast transmission games in wireless ad-hoc networks induced by different cost-sharing methods. We have given positive and negative existential proofs, bounded the price of anarchy and shown the intractability of determining good equilibria.

Unfortunately, even if not explicitly claimed, the rate of convergence for the only convergent method, the Shapley value, is exponential, unless specific choices for the players are selected. However, it is still open the determination of choices

yielding a convergence in a polynomial number of steps. To this aim, the strategy in which at each step a player chooses the best possible path is a good candidate. The answer to this question would also solve another interesting open problem, that is the determination of any Nash equilibrium, even non optimal.

Are there convergent cost sharing methods yielding a better price of anarchy?

Finally, it would be worth to extend all our results to particular cases, like the one of stations lying in Euclidean spaces with transmission costs induced by their distances.

References

1. V. Bilò, C. Di Francescomarino, M. Flammini, and G. Melideo. Sharing the cost of multicast transmissions in wireless networks. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2004.
2. V. Bilò and L. Moscardelli. The price of anarchy in all-optical networks. In *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2004.
3. A.E.F. Clementi, H. Huiban, P. Penna, G. Rossi, and P. Vocca. Some recent theoretical advances and open questions on energy consumption in static ad-hoc wireless networks. In *Proceedings of the 3rd International Workshop on Approximation and Randomization (ARACNE)*, pages 23–38, 2002.
4. V. Conitzer and T. Sandholm. Complexity results about nash equilibria. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 765–771, 2003.
5. E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to nash equilibria. In *Proceedings of the 30th Annual International Colloquium on Automata, Languages and Programming (ICALP)*. Lecture Notes in Computer Science, Springer-Verlag, 2003.
6. A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–351, 2003.
7. A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure equilibria. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing (STOC)*, 2004.
8. J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Approximation and collusion in multicast cost sharing. In *Proceedings of the 4th ACM Conference on Electronic Commerce (EC)*. ACM, 2003.
9. J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Hardness results for multicast cost sharing. *Journal of Public Economics*, 304(1-3):215–236, 2003.
10. J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proceedings of 32nd ACM Symposium on Theory of Computing (STOC)*, pages 218–227. ACM, 2000.
11. D. Fotakis, S. Koutogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 123–134, 2002.
12. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

13. K. Jain and V.V. Vazirani. Applications of approximation algorithms to cooperative games. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 364–372. ACM, 2001.
14. E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *LNCS*, pages 387–396, 1999.
15. M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 510–519, 2001.
16. I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13:111–124, 1996.
17. J. F. Nash. Equilibrium points in n -person games. In *Proceedings of the National Academy of Sciences*, volume 36, pages 48–49, 1950.
18. P. Penna and C. Ventre. Sharing the cost of multicast transmissions in wireless networks. In *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2004.
19. T.S. Rappaport. *Wireless communications: principles and practice*. Prentice-Hall, Englewood Cliffs, NY, 1996.
20. R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
21. T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of ACM*, 49(2):236–259, 2002.
22. L.S. Shapley. The value of n -person games. *Contributions to the theory of games*, pages 31–40, Princeton University Press, 1953.
23. A. Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 416–425, 2002.
24. J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 585–594. IEEE Computer Society, 2000.

Structural Similarity in Graphs^{*}

A Relaxation Approach for Role Assignment

Ulrik Brandes and Jürgen Lerner^{**}

Department of Computer & Information Science, University of Konstanz

Abstract. Standard methods for role assignment partition the vertex set of a graph in such a way that vertices in the same class can be considered to have equivalent roles in the graph. Several classes of equivalence relations such as regular equivalence and equitable partitions have been proposed for role assignment, but they all suffer from the strictness of classifying vertices into being either equivalent or not. It is an open problem how to allow for varying degrees of similarity. Proposals include ad-hoc algorithmic approaches and optimization approaches which are computationally hard.

In this paper we introduce the concept of *structural similarity* by relaxation of equitable partitions, thus providing a theoretical foundation for similarity measures which enjoys desirable properties with respect to existence, structure, and tractability.

1 Introduction

Role assignment is an important tool in the analysis of, e.g., social networks and food webs. Given a graph one tries to assign roles to vertices by looking for partitions of the vertex set such that equivalent vertices can be considered to occupy the same kind of structural position [20].

Whether a vertex partition yields a “meaningful” role assignment is up to some notion of compatibility with the edges of the graph. In their seminal paper, Lorrain and White [18] proposed that vertices have the same role if they are *regularly equivalent*, i.e. have identical neighborhoods. This rather restrictive requirement has later been weakened in many ways (see [21, 12] for an overview).

Vertex partitions are well investigated in some areas of computer science or graph theory. E.g. *quotient* partitions in algebraic graph theory [15] serve to find *invariants* of graphs and to determine the spectrum of highly symmetric, huge graphs. *Equitable* partitions (also *strongly regular* or *strongly regular*) are used to determine equivalent states of finite state processes in the calculus of communicating systems [19]. Shortly, a vertex coloring is called *regular* (*strongly regular*) if equivalent vertices have the same (number of each) colors in their neighborhood.

However, practical applicability of these kinds of vertex partitions to social networks or other irregular graphs is severely limited. Firstly it is \mathcal{NP} -complete

^{*} We gratefully acknowledge financial support from Deutsche Forschungsgemeinschaft (DFG, grant Br 2158/1-2)

^{**} Corresponding author, lerner@inf.uni-konstanz.de

to decide whether a graph has a regular, or equitable, partition with a given quotient graph (see Theorem 7). Secondly, most graphs analyzed in the social sciences have small or even trivial automorphism groups, irregularly distributed vertex degrees, etc., so that they hardly ever admit a non-trivial equitable partition anyway, while tractable regular partitions are often trivial as well (e.g. the maximal regular equivalence of an undirected graph is simply the division into isolates and non-isolates). Finally, while it makes sense that small perturbations like adding or deleting single edges destroy or establish the equivalence of vertices in, e.g., finite state processes, this is counterintuitive in the case of social networks, which often contain measurement errors and are expected to display regular patterns only approximately. In the case of irregular graphs we expect that small perturbations cause vertices to be more or less ... but do not decide about equivalence or non-equivalence.

We summarize these remarks by observing that the real obstacle in applying role assignment methods to empirical graphs is not an inappropriate formulation of compatibility of equivalence and graph structure, but the equivalence relation itself. In social networks and other graphs in which many pairs of vertices are somehow related, but not exactly equivalent, we need a notion of ... of vertices, rather than equivalence.

Other attempts to introduce degrees of similarity have been unsatisfactory so far: Algorithmic approaches like REGE and CATREGE lack “a theoretical rationale for the measure of similarity produced” [7–p. 375], while optimization approaches (e.g. [3]) are computationally hard and suffer from the existence of many local optima.

We therefore propose the notion of ... as relaxations of equivalence relations. From a compatibility requirement generalizing equitability, we obtain the subset of ... similarities which displays highly desirable mathematical properties. Our main results are that the set of all structural similarities can be described compactly by the eigenvalue decomposition of the adjacency matrix and that, in this framework, a graph R is a quotient of a graph G if and only if the characteristic polynomial of R divides the characteristic polynomial of G . In particular, this implies an efficient algorithm for the role assignment problem.

This paper is organized as follows. Basic notation is provided in Sect. 2. In Sect. 3 we derive similarities as relaxations of equivalence relations and introduce the condition for similarities to be compatible with a graph’s structure. A characterization and some interesting properties of the set of edge-compatible similarities are given in Sect. 4. Computational issues are addressed in Sect. 5 and are illustrated on a well-studied dataset [14] in Sect. 6.

2 Preliminaries

2.1 Basic Notation

In this paper we consider ... graphs, possibly with ... or edge ... For background on graph theory see, e.g., [11].

We also need some definitions from linear algebra (see, e.g., [1]). In particular we need the notions of \dots vectorspaces, i.e. real vectorspaces \mathcal{V} supplied with an \dots or \dots $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}$, and \dots of \dots linear mappings in such spaces.

2.2 Graph Spaces

As in [4], we associate with a graph $G = (V, E)$ the \dots $\mathcal{V} := \mathcal{V}(G) := \{f : V \rightarrow \mathbb{R}\}$, that is the vectorspace of all real-valued functions on the vertex set. \mathcal{V} has \dots $(f_v)_{v \in V}$, where $f_v(v) := 1$ and $f_v(u) := 0$ if $u \neq v$. An element f_v of the standard basis can be associated with the vertex v . Further, we define $\alpha(G) : \mathcal{V} \rightarrow \mathcal{V}$ to be the endomorphism determined by the action of the adjacency matrix $A(G)$ on the standard basis. This immediately generalizes to weighted graphs. Finally, we get an inner product $\langle \cdot, \cdot \rangle$ on \mathcal{V} which is uniquely defined by $\langle f_v, f_v \rangle := 1$ and $\langle f_u, f_v \rangle := 0$, if $u \neq v$, for $u, v \in V$.

Definition 1. \dots G \dots $\mathcal{G}(G) = (\mathcal{V}(G), \alpha(G))$ \dots graph space \dots G \dots $u, v \in \mathcal{V}$ \dots weight \dots (u, v) \dots $w(u, v) := \langle u, \alpha(v) \rangle$

Conversely, suppose we are given a pair $\mathcal{G} = (\mathcal{V}, \alpha)$, where \mathcal{V} is a Euclidean vectorspace and $\alpha : \mathcal{V} \rightarrow \mathcal{V}$ a symmetric endomorphism. Then \mathcal{G} induces a complete, weighted graph $G = (V, E)$ by choosing an orthonormal basis of \mathcal{V} as the vertex set V , and defining the weight of an edge $e = \{u, v\}$ by $w(e) := \langle u, \alpha(v) \rangle$. Thus a graph space might be seen as a collection of graphs. We stress that different bases will lead to different, possibly non-isomorphic, graphs.

Two graph spaces are identical if and only if the underlying vectorspaces and endomorphisms are the same. But, as for graphs, it is reasonable to identify graph spaces which differ only by a renaming of the vertex space. If $G = (V, E)$ and $H = (W, F)$ are graphs then an \dots from G to H is a bijection $\phi : V \rightarrow W$ such that for all $u, v \in V$, $A(G)_{uv} = A(H)_{\phi(u)\phi(v)}$, i.e., the adjacency matrices are, after appropriate ordering of vertices, identical. In the following definition, we relax the condition that isomorphisms have to be bijections between discrete sets to the condition that they have to be vectorspace-isomorphisms.

Definition 2. \dots $\mathcal{G} = (\mathcal{V}, \alpha)$ \dots $\mathcal{H} = (\mathcal{W}, \beta)$ \dots isomor-
 phism \dots \mathcal{G} \mathcal{H} \dots $\varphi : \mathcal{V} \rightarrow \mathcal{W}$ \dots
 \dots $u, v \in \mathcal{V}$

$$\langle u, \alpha(v) \rangle = \langle \varphi(u), \beta(\varphi(v)) \rangle$$

\dots $\alpha = \varphi^T \beta \varphi$ \dots \mathcal{G} \mathcal{H} \dots isomorphic

Graph isomorphisms are special, more restrictive types of graph space isomorphisms.

3 Structural Similarities

Relaxing (boolean) equivalence, we introduce a (continuous) notion of similarity. It is derived from projections that relax the surjective mapping associated with an equivalence relation (i.e. which maps elements to their equivalence classes). We show that for each projection there is a unique associated similarity and vice versa. A natural generalization of equitability then ensures that similarities respect the structure of a graph.

Let us start by considering the discrete case. For an equivalence relation \sim on the vertex set V of a graph $G = (V, E)$, let $W := V / \sim$ be its set of equivalence classes. The associated surjective mapping $\phi : V \rightarrow W$, that maps vertices to their equivalence class, defines a binary $|W| \times |V|$ matrix P where for $v \in V, w \in W, P_{wv} = 1$ iff $\phi(v) = w$. Such a matrix P satisfies the equation $PP^T = \text{id}_W$, for boolean matrix multiplication.

Relaxations of such class mappings allow for the entries p_{wv} , instead of 1's ("v is in class w") and 0's ("v is not in class w"), real numbers ("v's degree of membership to w is $p_{wv} \in \mathbb{R}$ ").

Definition 3. $\mathcal{V} \rightarrow \mathcal{W}$ projection, $\pi : \mathcal{V} \rightarrow \mathcal{W}$ projection, $\pi\pi^T = \text{id}_{\mathcal{W}}$

These generalized mappings to classes suggest the following generalization of quotients of graphs to quotients of graph spaces. Considering the discrete case, let P be the (binary) characteristic matrix of a surjective mapping of the vertices of G onto the set W of equivalence classes. Then, following [15–Sect. 9.6], we obtain a (directed, weighted) graph $H = (W, F)$ (called the \dots of G modulo \sim) that is defined by its adjacency matrix $A(H) := (PP^T)^{-1}PAP^T$, where A is the adjacency matrix of G . This definition is motivated by the fact that for two classes, say $w, w' \in W, A(H)_{ww'}$ is the average number of edges between vertices in w and vertices in w' . The translation to projections is straight-forward:

Definition 4. $\mathcal{G} = (\mathcal{V}, \alpha) \rightarrow \mathcal{W}, \pi : \mathcal{V} \rightarrow \mathcal{W}$
 $\mathcal{W}, \pi \rightarrow \alpha \rightarrow \mathcal{W}, \beta : \mathcal{W} \rightarrow \mathcal{W}$
 $\beta := \pi\alpha\pi^T$
 $\mathcal{G}/\pi = (\mathcal{W}, \beta) \rightarrow \mathcal{W}, \pi$ quotient $\mathcal{G} \rightarrow \pi$

Note that, since α is symmetric, β is symmetric.

We are now ready to define similarities as relaxations of equivalence relations, such that similarities and projections are associated with each other just like equivalence and class membership relations. Reconsidering the discrete case, let $\phi : V \rightarrow W$ be a surjective mapping and P be the (binary) characteristic matrix of ϕ . The equivalence relation induced by ϕ has characteristic matrix $S = P^T P$, since two vertices u and v are equivalent, iff the corresponding columns of P have the 1 in the same row, iff $s_{uv} = 1$.

If we relax P to a projection π , then $\sigma := \pi^T \pi$ is \dots , i.e. $\sigma^T = \sigma$, and \dots , i.e. $\sigma^2 = \sigma\sigma = \sigma$ and these properties serve to define our relaxation of equivalence relations.

Definition 5. Let \mathcal{V} be a euclidian vectorspace and $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ a similarity. Then $\sigma^T = \sigma$ and $\sigma^2 = \sigma$.

An equivalence relation \sim induces a similarity represented by the matrix S_\sim , called here the *normalized matrix* of \sim , which is given by

$$(S_\sim)_{uv} := \begin{cases} 0 & \text{if } u \not\sim v \\ 1/c & \text{if } u \sim v \text{ and } c \text{ is the size of the equivalence class of } v \end{cases}$$

For example, the partition $\{\{1, 2, 3\}, \{4, 5\}\}$ of $\{1, \dots, 5\}$ has normalized matrix

$$S = \begin{bmatrix} \left[\begin{smallmatrix} 1 \\ 3 \end{smallmatrix} \right]_{3 \times 3} & [0]_{3 \times 2} \\ [0]_{2 \times 3} & \left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix} \right]_{2 \times 2} \end{bmatrix}$$

Note that, when multiplying normalized matrices, idempotency corresponds to transitivity. For a mapping σ , idempotency means that σ restricted to its image $\text{im } \sigma$ is the identity.

Like equivalence relations and their associated surjective mappings, similarities and projections are just two points of view of the same concept: Let \mathcal{V} and \mathcal{W} be two euclidian vectorspaces and $\pi : \mathcal{V} \rightarrow \mathcal{W}$ a projection. Then, $\sigma_\pi := \pi^T \pi$ is symmetric and idempotent, i.e., a similarity. Conversely, let \mathcal{V} be a euclidian vectorspace and $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ a similarity. Set $\mathcal{W} := \text{im } \sigma$. Then, $\pi_\sigma = (v \mapsto \sigma(v)) : \mathcal{V} \rightarrow \mathcal{W}$ is a projection.

The next theorem states that $\sigma \mapsto \pi_\sigma$ and $\pi \mapsto \sigma_\pi$ are mutually inverse (up to isomorphism).

Theorem 1. Let (\mathcal{V}, α) be a euclidian vectorspace and $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ a similarity. Let \mathcal{W} be a euclidian vectorspace and $\pi : \mathcal{V} \rightarrow \mathcal{W}$ a projection. Then $\sigma_\pi := \pi^T \pi$ is a similarity and $\pi_{\sigma_\pi} := \pi \alpha \pi^T$ is a projection. Let $(\mathcal{U}, \beta_\mathcal{U})$ be a euclidian vectorspace and $\pi' : \mathcal{U} \rightarrow \mathcal{W}$ a projection. Then $\beta_\mathcal{U} := \pi' \alpha \pi'^T$ is a similarity and $\pi'_{\beta_\mathcal{U}} := \pi' \beta_\mathcal{U} \pi'^T$ is a projection. Finally, $\pi_{\sigma_\pi} = \pi'_{\beta_\mathcal{U}}$ and $\pi'_{\beta_\mathcal{U}} = \pi_{\sigma_\pi}$.

For proving the second part, note that $\varphi := \pi|_{\mathcal{U}} = \pi \pi'^T$ is an isomorphism of the graph spaces (\mathcal{W}, β) and $(\mathcal{U}, \beta_\mathcal{U})$. The proof therefore follows from the definitions and properties of the mappings in question. \square

As a consequence of Theorem 1 we conclude that for each similarity there is a unique associated projection and vice versa.

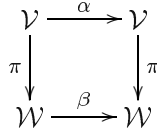
A similarity σ defines for a pair of vectors $u, v \in \mathcal{V}$ a real number $s(u, v)$, which is a measure for the similarity of u and v , by:

$$s(u, v) := \langle \pi(u), \pi(v) \rangle = \langle u, \pi^T \pi(v) \rangle = \langle u, \sigma(v) \rangle \quad ,$$

where $\pi := \pi_\sigma$. If u and v are basis vectors (vertices) then their similarity is the uv 'th entry of the matrix of σ .

Until now we have introduced similarities as relaxations of arbitrary equivalence relations. The following condition is a natural generalization of equitability and ensures that roles are based on the graph's structure.

Definition 6. Let (\mathcal{V}, α) and (\mathcal{W}, β) be graphs. A map $\pi : \mathcal{V} \rightarrow \mathcal{W}$ is called a structural similarity if $\beta := \pi\alpha\pi^T$.



$$\pi\alpha = \beta\pi \quad \text{structural similarity} \iff \sigma_\pi \text{ structural similarity}$$

4 Properties of Structural Similarities

We present several equivalent characterizations of structural similarity and show that equitable partitions form a special, restrictive case. Moreover, we generalize the partial order on the set of equivalence relations to a partial order on the set of similarities and show that the set of structural similarities forms a sublattice of the lattice of all similarities.

4.1 Characterization

Structural similarities can be characterized in several ways. A subspace $\mathcal{U} \subset \mathcal{V}$ is called α -invariant if $\alpha(\mathcal{U}) \subset \mathcal{U}$.

Theorem 2. Let (\mathcal{V}, α) and (\mathcal{W}, β) be graphs. A map $\pi : \mathcal{V} \rightarrow \mathcal{W}$ is a structural similarity iff $\sigma := \sigma_\pi$ is a similarity and

$$\begin{array}{ll}
 \pi \text{ is surjective} & \ker \sigma = \ker \pi \\
 \sigma\alpha = \alpha\sigma & \text{im } \sigma = \text{im } \alpha
 \end{array}$$

Proof. Let $\beta := \pi\alpha\pi^T$. **“ \Rightarrow ”**: Multiplying $\sigma\alpha = \alpha\sigma$ with π from the left, we get $\pi\pi^T\pi\alpha = \pi\alpha\pi^T\pi$, which implies $\pi\alpha = \beta\pi$. **“ \Leftarrow ”**: Given π with $\pi\alpha = \beta\pi$ we get

$$\sigma\alpha = \pi^T\pi\alpha = \pi^T\beta\pi = \pi^T\pi\alpha\pi^T\pi = \sigma\alpha\sigma$$

Taking the transpose of this equation and using symmetry of α and σ we obtain $\alpha\sigma = \sigma\alpha\sigma$, hence $\sigma\alpha = \alpha\sigma$. **“ \Leftarrow ”**: since, for the symmetric endomorphism σ , $\text{im } \sigma$ is α -invariant iff $\ker \sigma$ is α -invariant. **“ \Rightarrow ”**: Assume that $\alpha(\ker \sigma) \not\subset \ker \sigma$. Then there is a $v \in \mathcal{V}$ such that $\sigma(v) = 0$ and $\sigma\alpha(v) \neq 0$. Then, $\sigma\alpha(v) \neq 0 = \alpha\sigma(v)$, whence $\sigma\alpha \neq \alpha\sigma$. **“ \Leftarrow ”**: Let $v \in \mathcal{V} = \ker \sigma \oplus \text{im } \sigma$. We have $v = u_1 + u_2$ with $u_1 \in \ker \sigma$ and $u_2 \in \text{im } \sigma$. Then, since $\alpha(u_1) \in \ker \sigma$ and $\alpha(u_2) \in \text{im } \sigma$ (by **“ \Leftarrow ”**), we have that $\sigma\alpha(v) = \alpha(u_2) = \alpha\sigma(v)$, whence $\sigma\alpha = \alpha\sigma$. \square

As a corollary, Theorem 2 yields a compact description of structural similarities by the eigenvalue decomposition of the adjacency matrix.

Corollary 1. (\mathcal{V}, α) is structural iff $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ is structural iff $\ker \sigma$ ($\text{im } \sigma$) is α -invariant.

By Theorem 2, σ is structural iff $\ker \sigma$ (or $\text{im } \sigma$) is α -invariant. If $\ker \sigma$ ($\text{im } \sigma$) is generated by eigenvectors, then $\ker \sigma$ ($\text{im } \sigma$) is obviously invariant. Conversely, if $\mathcal{U} := \ker \sigma$ ($\mathcal{U} := \text{im } \sigma$) is α -invariant, then α restricted to \mathcal{U} is a symmetric endomorphism on \mathcal{U} and, thus, \mathcal{U} admits a basis of eigenvectors. \square

It follows from Theorem 2 that structural similarities are indeed a relaxation of equitable partitions.

Theorem 3. $G = (V, E) \sim V$ is structural iff $S \sim G$ is structural iff $\mathcal{G}(G) \sim G$ is structural.

This is a consequence of [15–Lemma 9.3.2] and the equivalence of 1. and 4. in Theorem 2. \square

4.2 Lattice Structure

We recall the notion of a lattice (see, e.g., [16]). A partially ordered set L such that for each $a, b \in L$ there is a least upper bound and an greatest lower bound of a and b . A lattice L is called complete if suprema and infima exist for each subset $L' \subset L$.

In the next lemma we establish a connection between similarities and subspaces, which we use to define a partial order on the set of similarities.

Lemma 1. $\mathcal{U} \subset \mathcal{V}$ is a subspace iff $\sigma : \mathcal{V} \rightarrow \mathcal{V}$ is a similarity with $\mathcal{U} = \ker \sigma$.

Suppose σ is such a similarity. First we observe that, since σ has to be symmetric, $\text{im } \sigma$ is the orthogonal complement of $\ker \sigma = \mathcal{U}$, hence $\text{im } \sigma$ is determined by \mathcal{U} . Thus, each $v \in \mathcal{V}$ admits a unique representation $v = v_1 + v_2$, where $v_1 \in \text{im } \sigma$ and $v_2 \in \mathcal{U}$. Further, since σ is idempotent, it is the identity on $\text{im } \sigma$. Thus

$$\sigma(v) = \sigma(v_1 + v_2) = v_1, \tag{1}$$

which shows the uniqueness. On the other hand, (1) can be used to define a similarity σ , such that $\ker \sigma = \mathcal{U}$, which shows the existence. \square

By the above lemma, we get a bijection between the set of subspaces $\mathbb{U}(\mathcal{V}) := \{\mathcal{U} \subset \mathcal{V}\}$ and the set of similarities $\mathbb{S}(\mathcal{V}) := \{\sigma : \mathcal{V} \rightarrow \mathcal{V}; \sigma^T = \sigma \text{ and } \sigma^2 = \sigma\}$. Via this bijection $\mathbb{S}(\mathcal{V})$ can be supplied with a partial order \leq by

$$\sigma \leq \tau \iff \ker \sigma \subset \ker \tau, \text{ for } \sigma, \tau \in \mathbb{S}(\mathcal{V}).$$

If \sim_1 and \sim_2 are two equivalence relations on the vertex set V of a graph $G = (V, E)$ such that \sim_1 is finer than \sim_2 , then the corresponding normalized matrices define similarities σ_1 and σ_2 satisfying $\sigma_1 \leq \sigma_2$. Thus, the embedding of

equivalence relations on V into the set of similarities $\mathbb{S}(\mathcal{V}(G))$ is order-preserving and the above-defined partial order is a generalization of the order on the set of equivalence relations.

$\mathbb{U}(\mathcal{V})$ is a complete lattice, where the infimum is given by intersection and the supremum by the sum of subspaces. Thus, $\mathbb{S}(\mathcal{V})$ is a complete lattice, too. We show that the set $\mathbb{S}_\alpha(\mathcal{V})$ of structural similarities forms a sublattice of $\mathbb{S}(\mathcal{V})$:

Theorem 4. $(\mathcal{V}, \alpha) \quad \mathbb{S}_\alpha(\mathcal{V}) \subseteq \mathbb{S}(\mathcal{V})$

By Theorem 2 it suffices to show that the intersection of α -invariant subspaces is α -invariant and that the sum of α -invariant subspaces is α -invariant. Proving this is straightforward. \square

The above theorem implies (see Theorem 5) that, given a similarity σ , that is not necessarily structural, there is always a smallest structural similarity above σ and a biggest structural similarity below σ . Note, that this doesn't hold, neither for regular equivalence relations [9], nor for equitable partitions, which can easily be verified.

Definition 7. $(\mathcal{V}, \alpha) \quad \sigma : \mathcal{V} \rightarrow \mathcal{V} \quad \mathcal{V}$
 (structural hull) $\sigma, \tau \in \mathbb{S}(\mathcal{V}) \quad \sigma \leq \tau \quad \tau' \in \mathbb{S}_\alpha(\mathcal{V}), \sigma \leq \tau' \quad \tau \leq \tau'$
 $\tau : \mathcal{V} \rightarrow \mathcal{V}$ (structural interior) $\sigma, \tau \in \mathbb{S}(\mathcal{V})$
 $\tau \leq \sigma \quad \tau' \in \mathbb{S}_\alpha(\mathcal{V}), \tau' \leq \sigma \quad \tau' \leq \tau$

Theorem 5. $(\mathcal{V}, \alpha) \quad \mathbb{S}_\alpha(\mathcal{V}) \subseteq \mathbb{S}(\mathcal{V})$

We utilize the fact that $L_1 := \mathbb{S}_\alpha(\mathcal{V})$ is a complete sublattice of the complete lattice $L := \mathbb{S}(\mathcal{V})$. Let $\sigma \in L$ then $\inf\{\tau \in L_1 ; \sigma \leq \tau\}$ is the structural hull and $\sup\{\tau \in L_1 ; \tau \leq \sigma\}$ is the structural interior of σ . \square

5 Determining Structural Similarities

Corollary 1 yields a general procedure for determining structural projections: Select a subset of eigenvalues and associated eigenvectors of a graph and project its graph space onto the subspace generated by these eigenvectors. For example, the equitable partition of the graph in Fig. 1 corresponds to a structural similarity that is the projection onto the sum of eigenspaces corresponding to eigenvalues 3, 1 and -2 .

If no eigenvalue has multiplicity greater than one (or we demand that eigenspaces must not be divided into subspaces), then Corollary 1 implies that, for fixed k , all structural projections onto k -dimensional image spaces can be listed efficiently. As a consequence of [13], the corresponding task for discrete partitions, i.e. to list all, say, regular equivalences with exactly k classes, is \mathcal{NP} -hard.

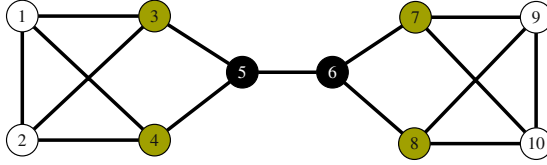


Fig. 1. Example from [6]. The vertex coloring represents an equitable partition of the drawn graph; the same partition can be found by a structural projection onto eigenvalues 3, 1, and -2

The following theorem characterizes role assignability with structural similarities.

Theorem 6. $\mathcal{G} = (\mathcal{V}, \alpha) \xrightarrow{\beta} \mathcal{R} = (\mathcal{W}, \beta) \xleftarrow{\alpha} \mathcal{U} \xrightarrow{\varphi} \mathcal{W}$
 $\pi : \mathcal{V} \rightarrow \mathcal{W} \quad \beta = \pi \alpha \pi^T$
 $\beta \quad \alpha$

The part can be proved along the same lines as [15–Theorem 9.3.3]. Let $\{(\lambda_1, \nu_1), \dots, (\lambda_r, \nu_r)\}$ be the set of different eigenvalues λ_i of β with multiplicity ν_i . For each $1 \leq i \leq r$ choose ν_i linearly independent eigenvectors $v_{i1}, \dots, v_{i\nu_i}$ of α with eigenvalue λ_i . Let $\mathcal{U} \subset \mathcal{V}$ be the subspace, generated by $(v_{11}, \dots, v_{1\nu_1}, v_{21}, \dots, v_{r\nu_r})$. Then, the projection $\pi_1 : \mathcal{V} \rightarrow \mathcal{U}$ is structural and $\beta_1 := \pi_1 \alpha \pi_1^T = \alpha|_{\mathcal{U}}$ has exactly the eigenvalues, with multiplicities, of β . Finally define $\varphi : \mathcal{U} \rightarrow \mathcal{W}$ to be the isomorphism which maps $(v_{11}, \dots, v_{r\nu_r})$ onto an orthonormal eigenbasis of (\mathcal{W}, β) in such a way that the image of v_{ij} has eigenvalue λ_i . The assertion follows with $\pi := \varphi \pi_1$. \square

The above theorem implies that the role assignment problem for structural similarities is easier than its discrete counterparts.

Theorem 7. $\mathcal{NP} \xrightarrow{G} \mathcal{R} \xrightarrow{R} \mathcal{G}(R)$
 $G \xrightarrow{R} \mathcal{R}$
 $G \xrightarrow{R} \mathcal{G}(R)$

1. is proved in [13]. 2. holds since the \mathcal{NP} -complete decision problem, whether a 3-regular graph has a perfect code [17], can be formulated as the problem whether this graph has an equitable partition with given quotient. The last statement follows immediately from Theorem 6. \square

6 Example: Southern Women Data

Theorem 6 is especially useful to determine structural projections. We illustrate this on a well-studied network representing 18 women who are connected

by weighted edges signifying the number of co-appearance at 14 selected social events (“Southern Women” data set [10]). A meta-analysis of this data is presented in [14].

It is a striking observation that a number of commonly used techniques can be seen as determining structural projections onto specific role graphs. The simple examples below mostly serve to illustrate that our method also gives a uniform interpretation to established, but seemingly unrelated methods of network analysis. It should be obvious that there are more sophisticated uses of Theorem 6 with more complex role graphs.

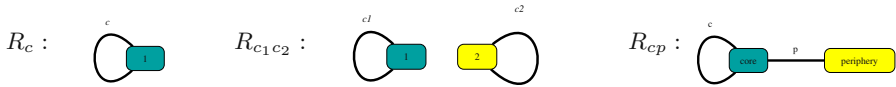


Fig. 2. Role graphs for centrality (*left*), 2-way partitioning (*middle*) and core/periphery structure (*right*)

A widely-used approach to determine the structural importance of vertices in a graph is eigenvector centrality [5], in which vertices are valued by the entries of the principal eigenvector of the adjacency matrix.

Vertices are therefore central if they have central neighbors, so that centrality can be viewed as the degree of membership in the only role present in role graph R_c shown in Fig. 2.

From the proof of Theorem 6 we know that a similarity σ is structural with $G/\sigma = R_c$, if and only if σ is the projection onto a 1-dimensional space generated by an eigenvector associated with eigenvalue $c > 0$ of the adjacency matrix of G . Therefore, eigenvector centrality is precisely a structural projection onto R_c , i.e. a 1-dimensional role assignment, and the eigenvalue determines the weight of the loop.

Eigenvectors are also frequently used to partition a graph into dense clusters. Using the spectrum of the adjacency matrix has been advocated, e.g., in [2]. Membership in a cluster can again be seen as a role, and partitioning into two roles thus corresponds to projecting onto role graph $R_{c_1 c_2}$ shown in Fig. 2.

Again, Theorem 6 implies that a similarity σ is structural with $G/\sigma = R_{c_1 c_2}$, if and only if σ is the projection onto a 2-dimensional space generated by eigenvectors corresponding to eigenvalues $c_1, c_2 > 0$. The result using the first two eigenvectors is shown in Fig. 3(.). This figure shows the usefulness of real-valued degrees of membership: Vertices 8 and 9 are rather between the two clusters, which is consistent with the fact that commonly used methods disagree largely about the cluster these vertices belong to [14]. Vertices 16, 17, and 18 are assigned to the second cluster, but only with low degrees of membership. Again, this is a precise result consistent with [14], where many methods did not assign these vertices at all.

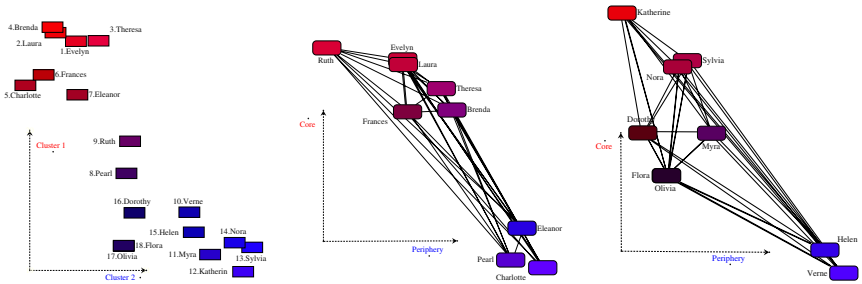


Fig. 3. projections for 2-clustering (*left*) and core/periphery structure in resulting clusters (*middle and right*)

Borgatti and Everett [8] discuss intuitive ideas and formal definitions for a frequently encountered phenomenon in social networks, namely their division into a cohesive ... of densely knit actors and a loosely connected ...

The degree of membership to both core and periphery of a vertex can be determined by a structural projection onto role graph R_{cp} of Fig. 2, although the spectrum of the role graph is not as trivial as those of the other two. However, the eigenvalues λ_1, λ_2 of R_{cp} satisfy $c = \lambda_1 + \lambda_2$ and $-p^2 = \lambda_1 \cdot \lambda_2$. Since it is reasonable to demand $c, p \in \mathbb{R}_{\geq 0}$, these equations imply w.l.o.g. $\lambda_1 \leq 0 \leq \lambda_2$ and $|\lambda_1| \leq |\lambda_2|$. Standard results in spectral graph theory imply that for loopless connected graphs the smallest and the largest eigenvalue always satisfy these inequalities. Thus, by Theorem 6, the projection σ onto the 2-dimensional space generated by the corresponding eigenvectors of G is structural with $G/\sigma = R_{cp}$. The resulting role assignment for the two clusters suggested by [14] of the data set is in Fig. 3 (... and ...). Observe that the more a vertex is in the core, the more it is connected to core and to peripheral vertices, whereas vertices that have high peripheral values are connected mostly to the core.

References

1. Artin, M.: Algebra. Prentice Hall (1991)
2. Barnes, E.R.: An algorithm for partitioning the nodes of a graph. SIAM Journal on Algebraic and Discrete Methods **3** (1982) 541–550
3. Batagelj, V., Doreian, P., Ferligoj, A.: An optimizational approach to regular equivalence. Social Networks **14** (1992) 121–135
4. Bollobás, B.: Modern Graph Theory. Springer (1998)
5. Bonacich, P.: Factoring and weighting approaches to status scores and clique identification. Journal of Mathematical Sociology **2** (1972) 113–120
6. Borgatti, S.P., Everett, M.G.: The class of all regular equivalences: Algebraic structure and computation. Social Networks **11** (1989) 65–88
7. Borgatti, S.P., Everett, M.G.: Two algorithms for computing regular equivalence. Social Networks **15** (1993) 361–376
8. Borgatti, S.P., Everett, M.G.: Models of core/periphery structures. Social Networks **21** (1999) 375–395

9. Boyd, J.P., Everett, M.G.: Relations, residuals, regular interiors, and relative regular equivalence. *Social Networks* **21** (1999) 147–165
10. Davis, A., Gardner, B., Gardner, M.: *Deep south*. The University of Chicago Press (1941)
11. Diestel, R.: *Graph Theory*. Springer-Verlag, New York (2000)
12. Everett, M.G., Borgatti, S.P.: Regular equivalence: General theory. *Journal of Mathematical Sociology* **19** (1994) 29–52
13. Fiala, J., Paulusma, D.: The computational complexity of the role assignment problem. In: *Proceedings of the ICALP 2003.*, Springer-Verlag (2003) 817–828
14. Freeman, L.C.: Finding social groups: A meta-analysis of the southern women data. In Breiger, R., Carley, K.M., Pattison, P., eds.: *Dynamic Social Network Modeling and Analysis*. The National Academies Press (2003)
15. Godsil, C., Royle, G.: *Algebraic Graph Theory*. Springer (2001)
16. Grätzer, G.: *General Lattice Theory*. Birkhäuser Verlag (1998)
17. Kratochvíl, J.: *Perfect Codes in General Graphs*. Academia Praha (1991)
18. Lorrain, F., White, H.C.: Structural equivalence of individuals in social networks. *Journal of Mathematical Sociology* **1** (1971) 49–80
19. Milner, R.: *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, 92. Springer Verlag, Berlin (1980)
20. Wasserman, S., Faust, K.: *Social Network Analysis: Methods and Applications*. Cambridge University Press (1994)
21. White, D.R., Reitz, K.P.: Graph and semigroup homomorphisms on networks of relations. *Social Networks* **5** (1983) 193–234

Flexibility of Steiner Trees in Uniform Orientation Metrics

Marcus Brazil¹, Pawel Winter², and Martin Zachariasen²

¹ ARC Special Research Centre for Ultra-Broadband Information Networks,
Department of Electrical and Electronic Engineering, The University of Melbourne,
Victoria 3010, Australia

brazil@unimelb.edu.au

² Department of Computer Science, University of Copenhagen,
DK-2100 Copenhagen Ø, Denmark

{pawel, martinz}@diku.dk

Abstract. We present some fundamental flexibility properties for minimum length networks (known as Steiner minimum trees) interconnecting a given set of points in an environment in which edge segments are restricted to λ uniformly oriented directions. These networks are referred to as λ -SMTs. They promise to play an increasingly important role in the future of optimal wire routing in VLSI physical design, particularly for the next generation of VLSI circuits. In this paper we develop the concept of a flexibility polygon for a λ -SMT, which is a region representing the union of all (minimum length) λ -SMTs with the same topology on a given set of points. We show that this polygon can be constructed, for a given point set and given topology, in linear time. We discuss some of the future applications of this polygon, which can be thought of as a geometric representation of the amount of flexibility inherent in a given λ -SMT.

1 Introduction

Interconnects in VLSI design have traditionally used rectilinear (or Manhattan) routing, in which only two perpendicular wiring directions are allowed. Recent technological advances in microchip fabrication have seen an increasing interest in the use of non-rectilinear interconnect architectures in VLSI design. The two alternative architectures that have generated the most interest in recent years are the Y-architecture [4], in which there are three directions for interconnects differing by angles of $2\pi/3$, and the X-architecture [9, 11], in which the rectilinear architecture is supplemented by the pervasive use of diagonal interconnects (at an angle of $\pi/4$ to the rectilinear interconnects). Both traditional rectilinear routing and these new proposed architectures are examples of so-called λ -geometry, in which a fixed set of $\lambda \geq 2$ uniformly oriented directions are allowed.

In VLSI routing, one of the principal objectives is to minimize the total length of interconnections in a net, that is, to compute a λ -geometry (or λ -SMT). This is in general an NP-hard problem. However, a powerful

exact algorithm, GeoSteiner, has recently been developed for this problem which can find an optimal solution for hundreds of randomly distributed points for arbitrary λ [5]. The key to this algorithm is to exploit the strong geometric structural properties of λ -SMTs. A recent paper [3] has established canonical forms for λ -SMTs which has led to a further speed-up of GeoSteiner.

In this paper we use these properties to explore the notion of flexibility in a λ -SMT. Informally, this is a measure of the extent to which edges in the minimum length network can be perturbed without increasing the length of the network. This has important applications in solving multi-objective optimisation problems in VLSI physical design, involving minimising the negative effects of properties such as congestion or signal delay as a secondary objective. The concept was introduced for rectilinear Steiner trees in [1] and [7]. Here we provide an effective measure of flexibility for a much wider class of minimum length networks, by defining and constructing the *flexibility polygon* of a λ -SMT; this polygon is a region representing the union of all (minimum length) λ -SMTs with the same topology on a given set of points. Some important applications of this concept in VLSI design are outlined in the concluding section of this paper.

2 Properties of λ -SMTs

We begin by establishing some basic definitions and notation, and reviewing some important properties of λ -SMTs. For a more detailed discussion of these properties, see [2] and [3].

Let $\lambda \geq 2$ be a given integer. Given λ orientations $i\omega$ ($i = 1, 2, \dots, \lambda$) in the Euclidean plane, where $\omega = \pi/\lambda$ is a unit angle, we represent these orientations by the angles with the x -axis of corresponding straight lines. A line or line segment with one of these orientations is said to be in a *legal direction*. Objects composed of line segments in legal directions are said to belong to a λ -*geometry*.

Since a minimum length network is necessarily a tree, we will only discuss networks in λ -geometry that are trees. We define a λ -tree to be a tree network in λ -geometry interconnecting a given set of points N , also denoted $\{N\}$. A λ -tree can contain nodes of degree 3 or more that are not terminals. These nodes are called *Steiner points*. Together the terminals and Steiner points are referred to as the *vertices* of the λ -tree.

The graph structure (i.e., pattern of adjacencies of the vertices) of a λ -tree is referred to as its *topology*. In this paper we are concerned with λ -trees T whose total edge length is minimum for a given set of terminals N ; these are the λ -*shortest Steiner trees* (λ -SMTs). If the total edge length of T is locally minimum, in that it is minimum for a given topology \mathcal{T} , then we say that T is a λ -SMT for \mathcal{T} .

Any λ -SMT T can be decomposed into a union of full subtrees meeting only at terminals. These subtrees are referred to as the *full Steiner trees* (FSTs) of T . A λ -SMT T for N is *flexible* if the number of FSTs is maximized over all λ -SMTs for N . In particular, for a full topology \mathcal{T} , a λ -SMT T for N and \mathcal{T} that is *flexible* cannot be replaced by two or more FSTs with the same total

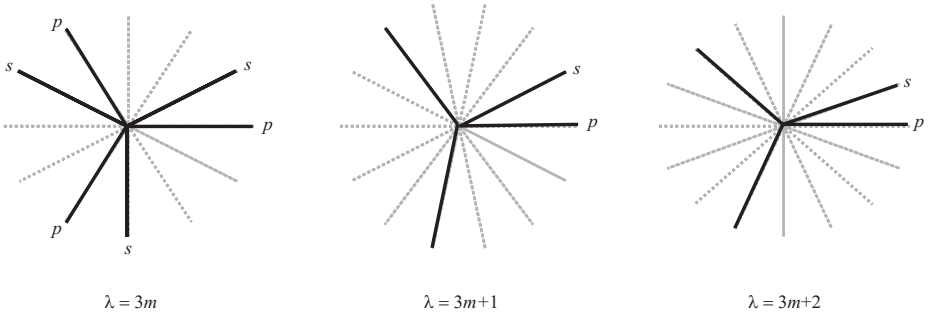


Fig. 1. The feasible directions (up to a rotation by a multiple of ω) for edges in a full λ -SMT for $m = 2$. Exclusively primary and secondary directions are indicated by p and s respectively

length (and, hence, fulsomeness is in a sense a property of N and \mathcal{T}). For any set of terminals there always exists a λ -SMT in which every full Steiner tree is fulsome. As in [3], we will focus our attention on full and fulsome λ -SMTs, which form the building blocks for all λ -SMTs. For a set of terminals N and a full topology \mathcal{T} for N , we denote by $S(N, \mathcal{T})$ the set of all full and fulsome λ -SMTs (or FSTs) interconnecting N and having topology \mathcal{T} . In view of the above, in the remainder of the paper we assume that $S(N, \mathcal{T}) \neq \emptyset$.

In [2, 8] it was shown that all Steiner points in a λ -SMT have degree 3 or 4. Furthermore, if a Steiner point with degree 4 exists in the λ -SMT, then it spans exactly four terminals (it is a cross); such a cross has no flexibility whatsoever, since any movement of the Steiner point increases the length of the tree [3]. Thus we will assume throughout this paper that every Steiner point has degree 3. The topology of any λ -SMT with degree 3 Steiner points is referred to as a λ -SMT with degree 3 Steiner points.

Edges in a λ -SMT are geodesics (in λ -geometry) between their endpoints. We refer to such an edge as a *straight edge* if it is a single straight line segment (in a legal direction), or else as a *bent edge* if it consists of two or more straight line components. It has been shown in [10] that bent edges are composed of line segments in exactly two legal directions differing by an angle of ω . Furthermore, although there are infinitely many ways of embedding a bent edge pq in the Euclidean plane, there are only two embeddings composed of exactly two straight line segments. The straight line components in such an embedding are referred to as *half-edges* and their intersections as *vertices*.

We now consider some important edge-direction properties in full λ -SMTs. In [3] it was shown that the straight edges and half-edges in a full λ -SMT can be oriented and then partitioned into three equivalence classes, such that each equivalence class contains oriented line segments in at most two directions differing by an angle of ω . In each equivalence class, the right-most edges (or half-edges) are labelled as *right-most edges*, and the left-most ones as *left-most edges*. Two important properties of this labelling are as follows:

- If λ is a multiple of 3 then there are exactly two feasible directions in each equivalence class. If λ is a not multiple of 3 then one of the equivalence classes contains two feasible directions and the other two classes each contain only one feasible direction, which is said to be both primary and secondary. A primary edge (or half-edge) that is not secondary is said to be *primary*. Similarly, a secondary edge (or half-edge) that is not primary is said to be *secondary*. This is illustrated for $\lambda = 6, 7$ and 8 in Figure 1.
- If λ is a multiple of 3 then two exclusively primary or exclusively secondary edges meet at a Steiner point at an angle of $2\pi/3$. If λ is a not multiple of 3 then any pair of edges meet at a Steiner point at an angle that differs from $2\pi/3$ by no more than ω .

Primary and secondary edges of a λ -SMT play a crucial role in determining flexibility. This is due to their connection with the 0-shifts of the paths in the tree, which we define below.

We define a *shift* of a straight edge pq to be a move of p to a new point $p' \neq p$ and a simultaneous move of q to a new point $q' \neq q$ such that $p'q' \parallel pq$. Similarly, a *shift* of a bent edge pq is defined to be a move of p to p' and a simultaneous move of q to q' such that $p'q'$ is either a bent edge with components in the same directions as those in pq or a straight edge whose direction is the same as that of one of the components of pq . The concept of a shift can be generalised to a path P in a full λ -SMT T as follows. A *shift* of $P = ps_1s_2 \dots q$ is a perturbation of T that moves the internal Steiner points s_i of P to s'_i (and fixes all other nodes of T) such that the following conditions are satisfied:

- (1) each internal Steiner point s_i of P moves along the line through s_i containing the straight edge or half-edge of T incident to s_i not lying on P ; and
- (2) the shift of P induces a shift on each internal edge of P .

Note that the effect of a shift is that it does not change the direction of any straight edge of T except possibly the first and last edges of P .

Given a subpath of a full λ -SMT T , we define a *shift* on that subpath to be a 0-shift if the shift does not increase the length of T . One of the key results on 0-shifts proved in [3] is the following.

Proposition 1. *Let T be a full λ -SMT with internal Steiner points s_1, s_2, \dots, s_n and edges e_1, e_2, \dots, e_m . Let P be a subpath of T containing the Steiner points s_1, s_2, \dots, s_k and edges e_1, e_2, \dots, e_l . Then a 0-shift of P exists if and only if $\lambda \geq 3k$.*

We define a *fundamental 0-shift* to be a (non-trivial) 0-shift that moves as few Steiner points as possible. If $\lambda = 3m$ (for some positive integer m) then a fundamental 0-shift moves one Steiner point; if $\lambda = 3m + 1$ or $3m + 2$ then a fundamental 0-shift moves two adjacent Steiner points. The fundamental 0-shifts are illustrated in Figure 2.

Theorem 1. *Let T be a full λ -SMT with internal Steiner points s_1, s_2, \dots, s_n and edges e_1, e_2, \dots, e_m . Let P be a subpath of T containing the Steiner points s_1, s_2, \dots, s_k and edges e_1, e_2, \dots, e_l . Then a fundamental 0-shift of P exists if and only if $\lambda \geq 3k$.*

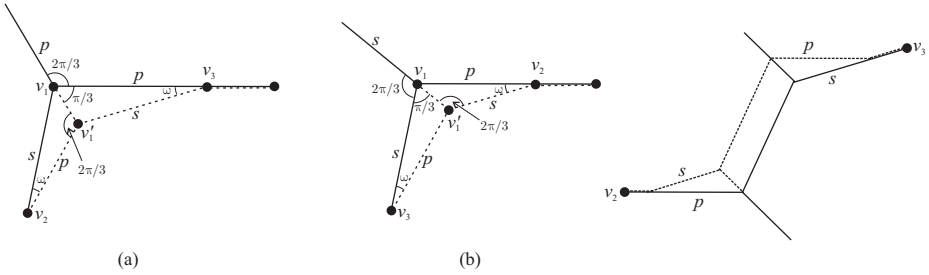


Fig. 2. The fundamental 0-shifts for $\lambda = 3m$ (cases (a) and (b)) and for $\lambda \neq 3m$. Exclusively primary and exclusively secondary edges are labelled p and s respectively

Consider the set $S(N, \mathcal{T})$ of full and fulsome λ -SMTs for a terminal set N and full Steiner topology \mathcal{T} . For $T \in S(N, \mathcal{T})$, define $p(T)$ to be the sum of the lengths of all exclusively primary edges in T and $s(T)$ to be the sum of the lengths of all exclusively secondary edges in T . We refer to $p(T)$ and $s(T)$ as, respectively, measures of the amount of exclusively primary material and exclusively secondary material in T . The following theorem shows that $p(T)$ and $s(T)$ depend only on N and \mathcal{T} .

Theorem 2. For $T \in S(N, \mathcal{T})$, $p(T) = \lambda \cdot p(N, \mathcal{T})$ and $s(T) = \lambda \cdot s(N, \mathcal{T})$.

3 The Flexibility Polygon and Its Properties

In this section we establish the basic properties of the flexibility polygon, a geometric object which allows us to determine the degree of flexibility available in a given full and fulsome λ -SMT. This polygon indicates the extent to which edges in the tree can be moved using 0-shifts without changing the length of the tree. More precisely, it places tight bounds on the possible positions of the edges of the trees in $S(N, \mathcal{T})$, the set of full and fulsome λ -SMTs for a given terminal set N and full Steiner topology \mathcal{T} . Recall that we assume that $S(N, \mathcal{T}) \neq \emptyset$; furthermore, we assume, without loss of generality, that $|N| \geq 3$ and that every Steiner point in \mathcal{T} has degree 3. Note that all λ -SMTs in $S(N, \mathcal{T})$ use the same set of edge directions, that is, every tree can be obtained from another tree using 0-shifts [3].

The flexibility polygon $F(N, \mathcal{T})$ for terminal set N and topology \mathcal{T} is defined to be the union of the embeddings of all λ -SMTs in $S(N, \mathcal{T})$. We will show that this union forms a simply connected region with a polygonal boundary whose vertices include the terminals of T . Examples of flexibility polygons for $\lambda = 4$ and $\lambda = 6$ are given in Figure 3. Notice that in some cases parts of the

flexibility polygon may degenerate into single edges (indicating that some edges may exhibit no flexibility at all).

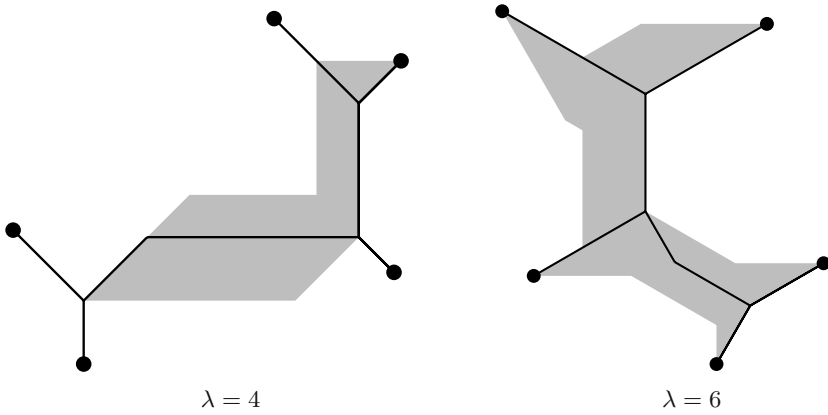


Fig. 3. Examples of λ -SMTs and flexibility polygons

Consider a counter-clockwise outer walk of T , beginning and ending at the same terminal. This walk allows us to place a cyclic ordering, $t_1 (= t_{n+1}), t_2, \dots, t_n$ on the terminals of T . We define the *rightmost concave path* in T to be the paths between t_i and t_{i+1} (for $i = 1, \dots, n$). In other words, these are paths between terminals where at each intermediate Steiner point we leave using the rightmost outgoing edge. Clearly, the set of all concave paths of T contains every edge of T exactly twice — once in each direction. In fact, up to the starting terminal, the order in which edges of the tree are visited by this outer walk of the tree is unique for a given terminal set N and topology \mathcal{T} ; this holds since all λ -SMTs in $S(N, \mathcal{T})$ can be obtained from each other using 0-shifts which do not change the ordering of the edges meeting at Steiner points [3].

In this section we will show that, intuitively, the flexibility polygon $F(N, \mathcal{T})$ can be constructed by pushing every concave path in T as far to the right as possible using 0-shifts. The resulting paths are referred to as *rightmost concave paths*, and in Theorem 4 it is shown that the union of all such paths is the boundary of $F(N, \mathcal{T})$.

We begin by studying the flexibility of edges and paths in T . Consider some Steiner point or corner point w in a λ -SMT T . Let T_w be one of the maximal subtrees of T having w as a leaf. We say that T_w is a *primary subtree* if all edges in T_w are primary edges, that is, use primary edge directions only. We define *secondary subtrees* analogously. Now we have the following result:

Lemma 1. *If T_w is a primary subtree of T and uv is an edge of T_w (with w at u), then $u'v'$ is an edge of T for any λ -SMT N in \mathcal{T} that is rightmost with respect to u and v (with w at u).*

A similar result holds if T_w is a λ -SMT subtree of T ; in this case no part of $u'v'$ is to the λ -left of the oriented line through u and v .

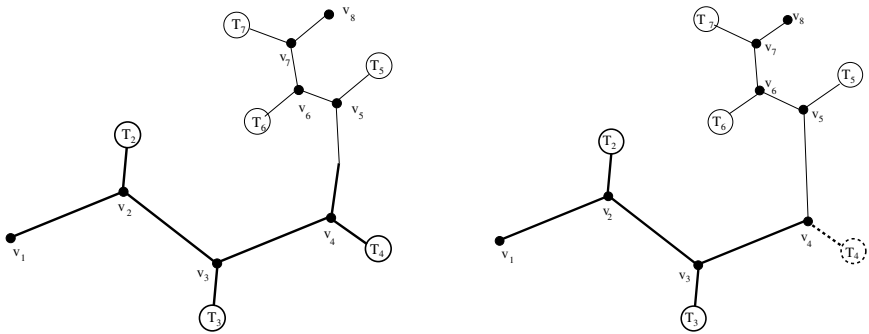


Fig. 4. The path P^r and its associated subtrees T_i . On the left is the case where the bent edge of P^r lies on P^r , and on the right the case where the bent edge does not lie on P^r .

Consider a path $P = v_1v_2 \dots v_{k-1}v_k$ connecting two vertices v_1 and v_k in T . For $i = 2, \dots, k-1$, let T_i denote the maximal subtree of T rooted at Steiner point v_i and not containing any edges of P (see Figure 4). We number the m edges of T by making a depth-first traversal from v_1 . At every Steiner point v_i the subtree T_i is traversed before the edge v_iv_{i+1} is traversed. We call this a depth-first traversal from v_1 along P . Note that the numbering of the edges in P depends only on the topology \mathcal{T} . Let T^r be the λ -SMT having the distribution of primary and secondary edges that results from this numbering $1, \dots, m$ of the edges [3]; this means that there exists an integer k , with $1 \leq k \leq m$, such that all edges numbered less than k are primary, and all edges numbered greater than k are secondary. The edge numbered k is the (possibly) bent edge.

Let $P^r = v_1v_2^r \dots v_{k-1}^rv_k$ be the path in T^r from v_1 to v_k . As a consequence of Theorem 2, P^r is uniquely defined for any depth-first traversal of T from v_1 along P . In other words, the coordinates of the vertices of P^r do not depend on the choice of depth first traversal in each of the subtrees T_i . We say that P^r is the rightmost path from v_1 to v_k .

Theorem 3. Let $(\lambda, \mu) \in \mathbb{R}^2$ be a point in the interior of the region \mathcal{R} . Let P^r be the rightmost path from v_1 to v_k in T^r . Let $u'v'$ be the line segment from u' to v' in $S(N, T)$. Let u and v be the endpoints of the line segment uv in $S(N, T)$. Then $u'v'$ is to the λ -left of the oriented line through u and v if and only if $u'v'$ is to the μ -left of the oriented line through u and v .

The shape of a rightmost path depends on the location of the bent edge in T^r . If the bent edge is located on P^r , say on edge $v_i^rv_{i+1}^r$, as in Figure 4 (left), all subtrees T_2^r, \dots, T_i^r will clearly be primary subtrees while all subtrees T_{i+1}^r, \dots, T_k^r will be secondary subtrees. Alternatively, if the bent edge is located in some subtree T_i , all edges in P^r are straight edges, all subtrees T_2^r, \dots, T_{i-1}^r

primary subtrees, and all subtrees T_{i+1}^r, \dots, T_k^r secondary subtrees (see Figure 4, right). In both cases there exists on P^r a node w , which may be either a corner point or Steiner point, such that the subtree of T having w as root and containing v_1 is \dots , while the subtree containing v_k is \dots . The theorem follows by applying Lemma 1. \square

The rightmost path of a \dots path is called a \dots . We now show that the flexibility polygon $F(N, \mathcal{T})$ can be described in terms of the rightmost concave paths for N and \mathcal{T} .

Theorem 4. \dots N \dots \mathcal{T} \dots N
 \dots $F(N, \mathcal{T})$ \dots N \dots \mathcal{T}

It immediately follows from Theorem 3 that the outer boundary of $F(N, \mathcal{T})$ is the union of the rightmost concave paths for N and \mathcal{T} .

Let T be a λ -SMT such that $T \in S(N, \mathcal{T})$. To see that $F(N, \mathcal{T})$ is simply connected, consider, for each pair of terminals t_i, t_{i+1} which are adjacent with respect to boundary order, the region $F^r(P)$ between the concave path $P = t_i \dots t_{i+1}$ of T and the corresponding rightmost concave path P^r . The region $F^r(P)$ is enclosed by the closed curve formed by P and P^r . We will now show that there exists a λ -SMT in $S(N, \mathcal{T})$ that intersects every point in $F^r(P)$. Clearly, this holds for all points on the boundary, that is, which are on either on P or P^r .

Therefore, consider a point p in the interior of $F^r(P)$. A sequence of 0-shifts that transforms P to P^r can be considered to be a continuous “contraction” of the closed curve given by P and P^r . Since p is not contained in the region obtained (which is P^r), there must be a point in time where the curve intersects p . Thus there exists a λ -SMT in $S(N, \mathcal{T})$ that intersects every point in $F^r(P)$.

The union of all regions $F^r(P)$ taken over all concave paths P of \mathcal{T} is the entire region bounded the rightmost concave paths for N and \mathcal{T} . \square

4 Construction of Flexibility Polygon

In this section we give a linear time algorithm for constructing the flexibility polygon $F(N, \mathcal{T})$ for a set of terminals N and a full Steiner topology \mathcal{T} . The algorithm to construct the flexibility polygon for N and \mathcal{T} consists of three steps, each of which can be performed in $O(\lambda n)$ time (where $n = |N|$). In the following we describe each of these steps in detail.

Step 1: Construction of a λ -SMT for N and \mathcal{T}

The first step of the algorithm is to construct an arbitrary λ -SMT $T \in S(N, \mathcal{T})$. This can be accomplished in $O(\lambda n)$ time using the algorithm given in [3]. Let $p(T)$ and $s(T)$ denote the total amounts of respectively exclusively primary and

exclusively secondary material in T ; by Theorem 2 $p(T)$ and $s(T)$ depend solely on N and \mathcal{T} — and not on the particular λ -SMT T .

Let $E(T)$ denote the set of edges (or arcs) in \mathcal{T} , and consider an edge $[u, v] \in E(T)$. All λ -SMTs in $S(N, \mathcal{T})$ use the same set of (at most) two edge directions for the edge $[u, v]$ [3]. By analysing the tree T , we obtain these (at most) two directions, corresponding to the the primary and secondary edge directions. Let $\Theta_p[u, v]$ and $\Theta_s[u, v]$ denote the primary and secondary edge directions for $[u, v]$, respectively. Note that we either have $\Theta_s[u, v] = \Theta_p[u, v] + \omega$ or $\Theta_s[u, v] = \Theta_p[u, v]$ (the latter is only possible when λ is not a multiple of 3).

Step 2: Construction of Primary and Secondary Subtrees

The second step of the algorithm is to construct so-called primary and secondary subtrees. For an edge $[u, v] \in E(T)$ the maximal subtree of topology \mathcal{T} rooted at u and not containing v is denoted by $\mathcal{T}[u, v]$. The primary subtree for $\mathcal{T}[u, v]$ (if it exists) is the embedding of $\mathcal{T}[u, v]$ such that every edge of $\mathcal{T}[u, v]$ uses its primary edge direction. In addition, we require that the amount of exclusively primary edge material $p[u, v]$ used by the primary subtree is less than $p(T)$, the total amount of exclusively primary material available. If the primary subtree for $\mathcal{T}[u, v]$ exists, we let $\Phi_p[u, v]$ denote the coordinates of the node u in this subtree; otherwise $\Phi_p[u, v] = \text{NIL}$.

The algorithm CONSTRUCTSUBTREES given in Figure 5 computes $\Phi_p[u, v]$ for every subtree $\mathcal{T}[u, v]$ in $O(n)$ time; note that there are $O(n)$ subtrees, since there are $O(n)$ oriented edges in $E(T)$. In the algorithm $(\Phi_p[u, v], \Theta_p[u, v])$ denotes the primary subtree with source $\Phi_p[u, v]$ and direction $\Theta_p[u, v]$. The function $d^*(q_1, q_2)$ returns the Euclidean distance between points q_1 and q_1 provided that the direction from q_1 to q_2 is an exclusively primary direction — otherwise $d^*(q_1, q_2)$ returns zero.

In the first phase of the algorithm (lines 1–10), we initialize $\Phi_p[u, v]$ for every subtree $\mathcal{T}[u, v]$. Also, every subtree $\mathcal{T}[u, v]$ such that u is a Steiner point with two terminals (other than v) as neighbours is inserted into a queue Q . The queue Q holds all subtrees $\mathcal{T}[u, v]$ that can be constructed at given point in time. A subtree $\mathcal{T}[u, v]$ can be constructed if the children of u in $\mathcal{T}[u, v]$ already have been constructed. Since any Steiner topology has at least one Steiner point with two neighbouring terminals, the queue is non-empty when the initialization phase finishes.

In the second phase of the algorithm (lines 11–23) we construct the subtrees that have been inserted into Q . The subtree is only constructed if the total amount of exclusively primary material is less than the total amount of exclusively primary material in T (lines 17–18). If the construction succeeds and v is a Steiner point, we investigate if the newly constructed subtree can be used to construct some larger subtree (lines 20–23). This is done by checking if either of the neighbours of v (other than u) already have had their subtree constructed; if this is the case then subtree rooted at v is inserted into Q .

Lemma 2. The algorithm CONSTRUCTSUBTREES runs in $O(n)$ time.

```

CONSTRUCTSUBTREES( $N, \mathcal{T}, T$ )
1 // Initialization phase
2  $Q = \emptyset$  // empty queue of oriented edges (=subtrees)
3 forall  $[u, v] \in E(\mathcal{T})$  do
4   if  $u \in N$  then
5      $\Phi_p[u, v] = u; p[u, v] = 0$ 
6   else
7      $\Phi_p[u, v] = \text{NIL}; p[u, v] = \infty$ 
8     Let  $v_1$  and  $v_2$  be the two neighbours of  $u$  other than  $v$ 
9     if  $v_1 \in N$  and  $v_2 \in N$  then
10      ENQUEUE( $Q, [u, v]$ ) //  $u$  has two neighbouring terminals
11 // Construction phase
12 while  $Q \neq \emptyset$ 
13    $[u, v] = \text{DEQUEUE}(Q)$ 
14   Let  $v_1$  and  $v_2$  be the two neighbours of  $u$  other than  $v$ 
15   Let  $r$  be the intersection (if any) between
16   the rays  $(\Phi_p[v_1, u], \Theta_p[v_1, u])$  and  $(\Phi_p[v_2, u], \Theta_p[v_2, u])$ 
17   if  $r$  exists then
18      $p[u, v] = p[v_1, u] + p[v_2, u] + d^*(\Phi_p[v_1, u], r) + d^*(\Phi_p[v_2, u], r)$ 
19     if  $p[u, v] < p(T)$  then
20        $\Phi_p[u, v] = r$  // subtree  $\mathcal{T}[u, v]$  has now been constructed
21       if  $v$  is a Steiner point then
22         Let  $u_1$  and  $u_2$  be the two neighbours of  $v$  other than  $u$ 
23         if  $\Phi_p[u_1, v] \neq \text{NIL}$  then ENQUEUE( $Q, [v, u_2]$ )
24         if  $\Phi_p[u_2, v] \neq \text{NIL}$  then ENQUEUE( $Q, [v, u_1]$ )

```

Fig. 5. Construction of primary subtrees of topology \mathcal{T}

The running time analysis is straightforward, since each oriented edge in $E(\mathcal{T})$ is inserted into Q at most once. Processing an edge (=subtree) takes $O(1)$ time.

The correctness follows by induction on the depth of the constructed subtrees. The base cases are the subtrees consisting of terminals only (that have depth 0), and Steiner points with terminals as children (that have depth 1); the latter are inserted into the queue Q in the initialization phase and therefore obviously constructed. For the induction step, assume that all subtrees with depth up to $k \geq 1$ have been constructed. It is then clear that subtrees of depth $k + 1$ will also be constructed, since these are inserted into Q when the smaller depth trees are constructed. \square

By using an analogous algorithm, we can also construct all secondary subtrees in $O(n)$ time. Here we let $\Phi_s[u, v]$ denote the coordinates of the node u in the secondary subtree $\mathcal{T}[u, v]$.

Step 3: Construction of the Boundary of the Flexibility Polygon

Consider the section of the flexibility polygon between a pair of consecutive terminals t_i and t_{i+1} (with respect to the terminal ordering from a counter-clockwise

outer walk of the tree). This is a rightmost concave path. It is constructed iteratively by building from t_i a path of edges of primary subtrees — constructed in step 2 — until we have a sequence of consecutive nodes u, v and w such that $\Phi_p[u, v] \neq \text{NIL}$ and $\Phi_p[v, w] = \text{NIL}$ (or $\Phi_p[u, v] \neq \text{NIL}$ and $v = t_{i+1}$). We distinguish between two cases:

- The secondary subtree $\mathcal{T}[v, u]$ exists (i.e., $\Phi_s[v, u] \neq \text{NIL}$). By Theorem 3 the intersection between the rays $(\Phi_p[u, v], \Theta_p[u, v])$ and $(\Phi_s[v, u], \Theta_s[v, u])$ must exist and defines a corner point of a bent edge. The boundary of the flexibility polygon consists of primary edges from t_i up to the corner point, and secondary edges from the corner point to t_{i+1} (Figure 4, left).
- The secondary subtree $\mathcal{T}[v, u]$ does not exist. By Theorem 3 the secondary subtree $\mathcal{T}[w, v]$ must exist. The intersection between the rays $(\Phi_p[u, v], \Theta_p[u, v])$ and $(\Phi_s[w, v], \Theta_s[w, v])$ defines the position of Steiner point v on the boundary of the flexibility polygon. Thus the boundary of the flexibility polygon consists of primary edges from t_i up to the Steiner point v , and secondary edges from v to t_{i+1} (Figure 4, right).

Therefore, by using the information computed in steps 1 and 2, we can construct the complete boundary of the flexibility polygon in one counter-clockwise outer walk of T .

Theorem 5. *The complete boundary of the flexibility polygon for a set of terminals N can be computed in $O(\lambda n)$ time, where $n = |N|$.*

5 Conclusion

The flexibility polygon is a compact description of the region in which λ -SMTs for a given set of terminals may be embedded. In VLSI routing a huge number of Steiner trees must be routed simultaneously on the surface of the chip. By placing the flexibility polygons for all nets on top of each other, congested regions are identified where many polygons overlap (Figure 6). Furthermore, this gives the basis for a completely new routing paradigm in VLSI routing (in arbitrary λ -geometry): Congested regions in the overlay graph associated with flexibility polygons give an indication of which nets are in congested regions. These nets should preferably be routed first in such a way that they avoid these regions.

Flexibility polygons have recently been used in a prototype of a new VLSI router for nets in λ -geometry [6]. Although computational results are still limited, and the router is only a prototype, the usability of flexibility polygons is remarkable when it comes to avoiding congested areas while keeping nets at minimal length with very few vias (changes of routing layers).

Using similar techniques, we have also been able to describe the regions in which individual terminals may be placed in a λ -SMT. This immediately gives flexibility regions for individual edges and bounds on their lengths under 0-shifts. We believe that these results will be important in improving delay-related measures for nets in VLSI routing. Finally, our result on the invariability

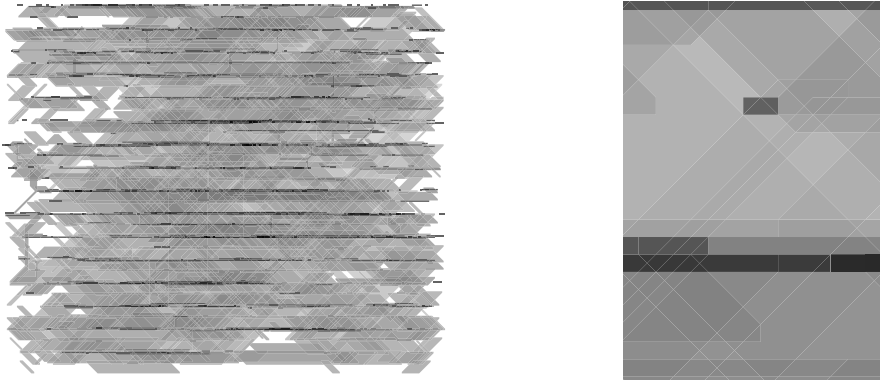


Fig. 6. Overlay of flexibility polygons in VLSI routing. On the left is a full picture of a small chip, and on the right a zoom-in of a region of the picture of the left. Darker regions indicate more congested regions

of exclusively primary/secondary edge material in a full and fulsome λ -SMT has the potential to improve existing exact algorithms for computing λ -SMTs.

Acknowledgments. This work was partially supported by a grant from the Australia Research Council and by a grant from the Danish Natural Science Research Council (51-00-0336).

References

1. E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh, Creating and exploiting flexibility in Steiner trees, *IEEE Trans. on Computer-Aided Design*, **22** (2003), 605-615.
2. M. Brazil, D. A. Thomas, and J. F. Weng, Minimum networks in uniform orientation metrics, *SIAM J. Comput.*, **30**(2000), 1579-1593.
3. M. Brazil, D. A. Thomas, J. F. Weng, and M. Zachariasen, Canonical forms and algorithms for Steiner trees in uniform orientation metrics. Tech. Rep. 02-22, Department of Computer Science, University of Copenhagen, 2002.
4. H. Chen, C. K. Cheng, A. B. Kahng, I. Mandoiu, Q. Wang and B. Yao, The Y-architecture for on-chip interconnect: Analysis and methodology, *Proc. IEEE/ACM Intl. Conf. Computer-Aided Design*, 2003, 13-19.
5. B. K. Nielsen, P. Winter, and M. Zachariasen, An exact algorithm for the uniformly-oriented Steiner tree problem, *LNCS 2461, Proceedings of the 10th European Symposium on Algorithms*, (2002), 760-772.
6. M. Paluszewski, P. Winter, and M. Zachariasen, A new paradigm for general architecture routing, *ACM Great Lakes Symp. on VLSI*, (2004), 202-207.
7. S. Peyer, M. Zachariasen, D. G. Jørgensen, Delay-related secondary objectives for rectilinear Steiner minimum trees. *Discrete Applied Mathematics*, **136**(2004), 271-298.

8. M. Sarrafzadeh and C. K. Wong, Hierarchical Steiner tree construction in uniform orientations, *IEEE Trans. on Computer-Aided Design*, **11**(1992), 1095–1103.
9. S. Teig, The X Architecture, *Proc. ACM/IEEE Workshop on System Level Interconnect Prediction*, (2002), 33–37.
10. P. Widmayer, Y. F. Wu, and C. K. Wong, On some distance problems in fixed orientations. *SIAM J. Comput.*, **16**(1987), 728–746.
11. The X Initiative, <http://www.xinititive.com>.

Random Access to Advice Strings and Collapsing Results

Jin-Yi Cai and Osamu Watanabe

¹ Computer Sci. Dept., Univ. of Wisconsin, Madison, WI 53706, USA
jyc@cs.wisc.edu

² Dept. of Math. and Comp. Sci., Tokyo Inst. of Technology
watanabe@is.titech.ac.jp

Abstract. We propose a model of computation where a Turing machine is given random access to an *advice string*. With random access, an advice string of exponential length becomes meaningful for polynomially bounded complexity classes. We compare the power of complexity classes under this model. It gives a more stringent notion than the usual model of computation with relativization. Under this model of random access, we prove that there exist advice strings such that the Polynomial-time Hierarchy PH and Parity Polynomial-time $\oplus P$ all collapse to P. Our main proof technique uses the decision tree lower bounds for constant depth circuits [Yao85, Cai86, Hås86], and the algebraic machinery of Razborov and Smolensky [Raz87, Smo87].

1 Introduction

In computational complexity theory, we cannot separate between many complexity classes. It is generally believed that these separation results are very hard to prove. Among the supporting evidence for such a pessimistic belief, people frequently cite the collapsing results under Δ_1^P , especially for complexity classes defined in non-randomized terms, such as P, NP, Σ_d^P , $\oplus P$, PSPACE, etc.

Consider, for example, the most famous P vs. NP conjecture. Baker, Gill and Solovay [BGS75] showed that we can relativize it in both ways. That is, there exist two oracles A and B such that $P^A = NP^A$ (the Δ_1^P holds) and $P^B \neq NP^B$ (the Δ_1^P does not hold). Intuitively, for each oracle set X , the relative computation model allowing oracle queries to X provides a “relativized complexity world” where all computation is the same as our real world except that one can use some special set of instructions, i.e., queries to the oracle set X . It is said that most of known proofs can be Δ_1^P ; that is, they are applicable in such relativized worlds. Therefore, having the above oracles A and B means that these proof techniques can not resolve the P vs. NP conjecture. For P vs. NP or PSPACE, perhaps the most straightforward proof of a relativized collapse is $P^{QBF} = NP^{QBF} = PSPACE^{QBF}$.

However, we feel that this argument is based on a model of computation which is not stringent enough. This is especially true for most of the relativized collapsing results. More precisely, relativized collapsing results are often proved by

allowing stronger usage of an oracle to a simulating machine than to a simulated machine.

Consider two complexity classes \mathcal{C}_1 (such as P) and \mathcal{C}_2 (such as NP or PSPACE). Let $\{\mathcal{M}_i\}$ be an enumeration representing the class \mathcal{C}_2 , and let \mathcal{M} be an arbitrary machine from this enumeration. A typical proof for a relativized collapsing result is to code the computation of \mathcal{M} for inputs of length n , in the oracle, in such a way that another machine \mathcal{M}' representing \mathcal{C}_1 can recover the results. In order not to “interfere” with computations of \mathcal{M} at length n , these results are coded at locations . . . what \mathcal{M} can access at input of length n , and \mathcal{M}' is allowed a running time and oracle access greater than that of \mathcal{M} . This encoding is sometimes explicitly carried out, sometimes implicitly done such as with the proof of $P^{\text{QBF}} = \text{PSPACE}^{\text{QBF}}$. In terms of the simulation by the P^{QBF} machine \mathcal{M}' simulating the $\text{PSPACE}^{\text{QBF}}$ computation \mathcal{M} on an input x , \mathcal{M}' will access an oracle location polynomially longer than where the corresponding access \mathcal{M} makes. That is, \mathcal{M}' is given more powerful oracle access than \mathcal{M} . One can argue that this asymmetry is within a polynomial factor, but it nonetheless denies access to certain segments of the oracle to the simulated machine while affords such access to the simulating machine. Moreover, if one actually relativizes the proofs of the few separation results such as the hierarchy theorems, one observes that this asymmetry is . . . present in the relativized proof.

In order to rectify this problem we propose a model of computation that is more stringent than the usual relativization computation. This turns out to be equivalent to a generalization of the notion of advice strings proposed by Karp and Lipton [KL80]. Intuitively, any relativized result can be regarded as a comparison between complexity classes under a certain nonuniform setting provided by an (infinite) advice, namely an oracle. Here we generalize the advice string formulation of Karp and Lipton by allowing random access to the advice string, so that advice strings longer than polynomial length become meaningful for polynomial time bounded computations. Then we compare complexity classes, given such nonuniform advice strings. That is, we compare two machines \mathcal{M}_1 and \mathcal{M}_2 (from complexity classes \mathcal{C}_1 and \mathcal{C}_2 respectively) that have random access to the same advice string s_n that is a priori given for their computation of any input of length n . Both machines will have complexity bounds that allow access to any bit of the advice string. This way we compare them on the same footing. Note that, since the advice string has a length accessible to both \mathcal{M}_1 and \mathcal{M}_2 , we cannot in general “preserve” the computation of one and let it be read by another, as in the usual relativization model.

Our main results in this paper show that both parity polynomial-time $\oplus\text{P}$ and the polynomial-time hierarchy PH collapse to P for some exponential-size advice strings. More precisely, for P and $\oplus\text{P}$ (resp., P and PH), we show some set $\{s_n\}_{n \geq 0}$ of advice strings of length $2^{(1+\epsilon)n}$, i.e., each s_n of length $2^{(1+\epsilon)n}$, with which $\oplus\text{P}$ (resp., PH) collapses to P. We use decision tree lower bounds for constant depth circuits [Yao85, Cai86, Hås86] and the algebraic machinery of Razborov and Smolensky [Raz87, Smo87]. It is open whether one can collapse PSPACE and P with some set of advice strings of some exponential size.

Results of this type are mainly of value in delineating the limit of our ability to settle some outstanding questions on complexity classes. Our model of random access to advice strings provides a more stringent model than the usual relativization model, and therefore it provides a more stringent perspective on the “provability” question. The open status of a collapse of PSPACE to P under random access to advice is particularly interesting in view of a result of Kozen [Ko78]: If $\text{PSPACE} \neq \text{P}$, then there exists a proof of this fact by diagonalization.

2 Random Access to Advice Strings

Recall the definition of \mathcal{C}/poly by Karp and Lipton [KL80]. We generalize this notion by allowing the underlying machines to have random access to an advice string. Let us fix any “length function” ℓ from \mathbf{N} to \mathbf{N} . A function $s : n \mapsto \{0, 1\}^{\ell(n)}$ is called an *advice string* of length $\ell(n)$. Given any advice function s of size $\ell(n)$, we say a language L is in the class \mathcal{C}/s if there is some machine \mathcal{M} representing the class \mathcal{C} , such that $x \in L$ iff $\mathcal{M}(x; s(|x|))$ accepts, where we denote the computation \mathcal{M} on x with random access to $s(|x|)$ by $\mathcal{M}(x; s(|x|))$. (The notion of \mathcal{C}/s is the usual one: A machine \mathcal{M} can write down an index to a bit of $s(|x|)$ on a special tape and then it gets that bit in unit time.) We denote this language as $L(\mathcal{M}; s)$. Clearly, if a time bound being considered is larger than the advice size, then the random accessibility is not necessary, and this notion is the same as the one by Karp and Lipton. (In the following, all complexity bounds and length functions are time and space constructible as appropriate. Furthermore, we assume that $\log(\ell(n))$ is polynomially bounded, which is reasonable for comparing with polynomial-time classes even if we allow random access to an advice string.)

Let s be any advice function. We say $\mathcal{C}_1/s \subseteq \mathcal{C}_2/s$ (write as $\mathcal{C}_1/s \subseteq \mathcal{C}_2/s$) if for every machine \mathcal{M}_1 representing \mathcal{C}_1 , there is a machine \mathcal{M}_2 representing \mathcal{C}_2 , such that $L(\mathcal{M}_1; s) = L(\mathcal{M}_2; s)$. We say $\mathcal{C}_1/s = \mathcal{C}_2/s$ (write as $\mathcal{C}_1/s = \mathcal{C}_2/s$) if both $\mathcal{C}_1/s \subseteq \mathcal{C}_2/s$ and $\mathcal{C}_2/s \subseteq \mathcal{C}_1/s$. On the other hand, we say $\mathcal{C}_1/s \not\subseteq \mathcal{C}_2/s$ (write as $\mathcal{C}_1/s \not\subseteq \mathcal{C}_2/s$) if there exists some machine \mathcal{M}_1 representing \mathcal{C}_1 such that $L(\mathcal{M}_1; s) \neq L(\mathcal{M}_2; s)$ for any machine \mathcal{M}_2 representing \mathcal{C}_2 .

Then our main results can be stated as follows.

Theorem 1. $\text{P}/s = \text{P} \iff \exists \delta > 0, \ell(n) \geq 2^{(1+\delta)n}$
 $\oplus \text{P}/s = \text{P}/s \iff \exists \delta > 0, \ell(n) \geq 2^{(1+\delta)n}$

Remark. $\text{P} \not\subseteq \text{Mod}_p \iff \exists \delta > 0, \ell(n) \geq 2^{(1+\delta)n}$

Theorem 2. $\text{PH}/s = \text{P}/s \iff \exists \delta > 0, \ell(n) \geq 2^{(1+\delta)n}$

3 Class P Versus Class $\oplus P$

In this section we consider the relation between P and $\oplus P$ and prove Theorem 1. The proof techniques will be extended in the next section to prove Theorem 2.

To simplify the presentation we will consider only $\log(\ell(n)) = (1 + \delta)n$. It is easy to extend the following proof to any $\ell(n)$ with $\log(\ell(n)) \geq (1 + \delta)n$.

Proof of Theorem 1. Let $\mathcal{M}_1, \mathcal{M}_2, \dots$ be a standard enumeration of all $\oplus P$ machines. Our goal is to construct an advice function s with $s(n) \in \{0, 1\}^{\ell(n)}$, with which the computation of every $\mathcal{M}_i(x; s(|x|))$ can be simulated by some P computation with the common advice $s(|x|)$. Let us fix any $\oplus P$ machine \mathcal{M} and any input length n , and discuss how to design $s(n)$ so that some P machine can simulate \mathcal{M} on $\{0, 1\}^n$ with advice $s(n)$. It would be easy later to “paste” together a single $s(n)$ for all machines to be considered at length n . (Only finitely many need to be dealt with at any finite length n . We will omit this detail.)

Let $m = n^{O(1)}$ be the maximum number of accesses to the advice string made by \mathcal{M} on any nondeterministic path on any input of length n . We assume that n is sufficiently large.

Let $L = 2^{(1+\delta)n}$. We will consider the advice string $s(n)$ of length L as being indexed by a binary string of length $I = (1 + \delta)n$.

For any $x \in \{0, 1\}^n$, define S_x to be some subset of $\{0, 1\}^I$ of size $\approx nm$. We want $\{S_x\}_{x \in \{0, 1\}^n}$ to be a family of pair-wise disjoint subsets of $\{0, 1\}^I$. For example, for $s = \lceil \log nm \rceil$, we can define

$$S_x = \{xu0^{I-(n+s)} \mid u \in \{0, 1\}^s\}.$$

Each string in $\bigcup_{x \in \{0, 1\}^n} S_x$ is the index of a bit in $s(n)$. We assign Boolean variables for these bits, and denote the set of these Boolean variables as Z . Let $M = |Z|$; note that $M \leq 2nm2^n \ll 2^I$. Let us name the Boolean variables in Z as z_1, z_2, \dots, z_M .

Assign arbitrarily the bit values for all bits in $s(n)$ other than those in Z . Then, for any input $x \in \{0, 1\}^n$, $\mathcal{M}(x; s(n))$ is completely determined by the values of z_i . That is, $\mathcal{M}(x; s(n))$ is a function on Boolean variables z_1, \dots, z_M . Furthermore, since $\mathcal{M}(x; s(n))$ is a parity computation asking at most m queries on each nondeterministic path, we may consider $\mathcal{M}(x; s(n))$ as a parity (or its negation) of (at most $\sum_{i=0}^m 2^i \binom{M}{i}$ many) conjunctions of at most m literals from z_1, \dots, z_M . Thus, $\mathcal{M}(x; s(n))$ is expressed by a polynomial $f_x(z_1, \dots, z_M)$ of degree $\leq m$ with integer coefficients mod 2. Note that $f_x(z_1, \dots, z_M)$ is multilinear, because we may assume that each bit is not queried more than once on each nondeterministic path.

Now we would like to assign z_1, \dots, z_M so that the following system of equations (*1) holds (under the mod 2 computation) for $\{0, 1\}^n = \{x_1, \dots, x_N\}$ (where $N = 2^n$).

$$(*1) \quad \begin{cases} f_{x_1}(z_1, \dots, z_M) = 1 - \prod_{z_j \in S_{x_1}} z_j, \\ \vdots \\ f_{x_N}(z_1, \dots, z_M) = 1 - \prod_{z_j \in S_{x_N}} z_j. \end{cases}$$

If this assignment is feasible (i.e., the advice string $s(n)$ is constructed satisfying (*1)), then for any $x \in \{0, 1\}^n$, one simply needs to check the membership of elements of S_x ; $\mathcal{M}(x; s(n))$ can then be computed as $1 - \prod_{z_j \in S_x} z_j$ in polynomial time.

Suppose, for a contradiction, that this is impossible to achieve. Then, since for every 0 or 1 value of z_1, \dots, z_M , each f_x takes a 0 or 1 value, it follows that for every assignment to the z_1, \dots, z_M , there exists some $x \in \{0, 1\}^n$ such that

$$f_x(z_1, \dots, z_M) = \prod_{z_j \in S_x} z_j.$$

Thus, for all 0,1-assignments to z_1, \dots, z_M , we have

$$\prod_{1 \leq i \leq N} \left[\prod_{z_j \in S_{x_i}} z_j - f_{x_i}(z_1, \dots, z_M) \right] = 0.$$

Then it follows from Fact 1 stated below that modulo the ideal $J = (z_1^2 - z_1, \dots, z_M^2 - z_M)$, the left hand side expression is identical to 0. In other words, we have the identity

$$\prod_{1 \leq i \leq N} \prod_{z_j \in S_{x_i}} z_j = L(z_1, \dots, z_M),$$

in the ring $\mathbf{Z}_2[z_1, \dots, z_M]/J$, where L is a polynomial of degree at most $(N - 1)2^s + m$. On the other hand, the degree of the lefthand side of the above equality is $N2^s$, which is larger than $(N - 1)2^s + m$. A contradiction. \square

Fact 1. $\dots p \dots F(x_1, \dots, x_n) \dots J = (x_1^2 - x_1, \dots, x_n^2 - x_n) \dots \mathbf{Z}_p[x_1, \dots, x_n]/J \dots F(x_1, \dots, x_n) \dots$

4 Class P Versus Class PH

We now show that there exists an advice function of advice size $2^{(1+\delta)n}$, such that the class PH collapses to P with random access to the advice strings given by the advice function. The modification in the proof from $2^{(1+\delta)n}$ to larger $\ell(n)$ is obvious. For simplicity of presentation we will assume $\ell(n) = 2^{(1+\delta)n}$ in what follows. We prove the following result for a fixed level Σ_d^P ; the construction for the advice string for PH follows since PH is a countable union of classes Σ_d^P , $d \geq 0$.

Theorem 3. $\dots d \geq 0, \dots \delta > 0, \dots \ell(n) = 2^{(1+\delta)n} \dots \Sigma_d^P/s = P/s$

Before stating our proof in detail, we explain its outline and some background. We begin by recalling the decision tree version of the Switching Lemma.

Some notions and notations first. For any Boolean function f over variables x_1, \dots, x_n , a random function ρ is a random function that assigns each x_i either 0, 1, or *, with probability $\Pr[\rho(x_i) = *] = p$ (for some specified parameter p) and $\Pr[\rho(x_i) = 0] = \Pr[\rho(x_i) = 1] = (1 - p)/2$, for each i independently. Assigning * means to leave it as a variable. Let $f|_\rho$ denote a function obtained by this random restriction.

The decision tree complexity of a Boolean function f , denoted by $DC(f)$, is the smallest depth of a Boolean decision tree computing the function. It can be shown easily that if $DC(f) \leq t$, then f can be expressed both as an AND of OR's as well as an OR of AND's, with bottom fan-in at most t . Moreover, what is crucial for our argument is the following property: If $DC(f) \leq t$, then f can be expressed as a polynomial on the variables, with integer coefficients and with degree at most t . In fact this polynomial always evaluates to 0 or 1, for any 0-1 assignments to its variables.

Superpolynomial lower bounds for constant depth circuits were first proved by Furst, Saxe and Sipser [FSS81], and by Ajtai [Ajt83]. Exponential lower bounds of the form $2^{n^{\Omega(1/d)}}$ for depth d circuits were first proved by Yao [Yao85] in a breakthrough result. Yao's bound was further improved by Håstad [Hås86] to $2^{\frac{1}{10}n^{\frac{1}{d-1}}}$, and his proof has become the standard proof. Independently, Yao's work was improved upon in another direction. Cai [Cai86] investigated whether constant depth circuits of size $2^{n^{\Omega(1/d)}}$ must err on an asymptotically 50 % of inputs against parity. To attack this problem, the decision tree point of view was first introduced in [Cai86]. This approach in terms of inapproximability has been found most fruitful in the beautiful work of Nisan and Wigderson [Nis91, NW94] on pseudorandom generators.

Adapting Håstad's proof to the decision tree model, one can prove the following.

Lemma 1. Let C be a circuit of depth $d + 1$ with $\text{size}(C) \leq t$. Let z_1, \dots, z_L be a set of L variables.

$$\Pr[DC(C|_\rho) \geq t] \leq \frac{\text{size}(C)}{2^t},$$

where ρ is a random restriction with $p = \Pr[z_i = *] = 1/(10t)^d$.

We now explain our construction. Fix any Σ_d^P machine \mathcal{M} and any sufficiently large input length n . We want to construct $s(n)$, such that the computation $\mathcal{M}(x; s(n))$ can be simulated by a polynomial-time deterministic machine, for all x of length n . Constructing the advice function s for the simulation of Σ_d^P machines can be done as before for $\oplus P$ and is omitted here.

Thus, from now on, we are concerned with the simulation of \mathcal{M} on 2^n inputs of length n . Let m be an integer bounding \mathcal{M} 's running time on inputs of length n , where $m = O(n^k)$ for some $k \geq 0$. Let $I = (1 + \delta)n$ and $L = 2^I$. Let

z_1, z_2, \dots, z_L be Boolean variables denoting the bits in $s(n)$. Let Z denote the set of all Boolean variables z_1, \dots, z_L . With a slight abuse of notation we will also let Z denote a set of corresponding indeterminants.

For any input string $x \in \{0, 1\}^n$, consider the computation of $\mathcal{M}(x; s(n))$. The computation $\mathcal{M}(x; s(n))$ is a function from the Boolean variables z_1, \dots, z_L to $\{0, 1\}$. Furthermore, since \mathcal{M} is a Σ_d^P machine, by a standard interpretation (see [FSS81]) of the Σ_d^P query computation, we may regard $\mathcal{M}(x; s(n))$ as a depth $d+1$ circuit on input variables z_1, \dots, z_L , of size at most $m2^m$ and bottom fan-in at most m .

Our first step is to assign a random restriction ρ to z_1, \dots, z_L of an appropriate probability $p_0 = \Pr\{z_i = *\}$. By Lemma 1, with high probability the circuit is reduced to small depth decision trees with depth $t = 2m$. In fact, by choosing p_0 appropriately, we can even show that with high probability, a random restriction converts \dots circuits for all 2^n input strings to depth t decision trees.

Then these small depth decision trees can be expressed by low degree (i.e., degree $2m$) polynomials with integer coefficients. That is, after the random restriction, each computation $\mathcal{M}(x; s(n))$ is expressed as a degree $2m$ polynomial p_x . We have arrived at a similar situation to the parity computation. We will use a similar technique to attack this. However the exact approach in the $\oplus P$ case does not work.

In the $\oplus P$ case the function encoded is essentially the AND function $\bigwedge z_j$. This will not survive the random restriction. Instead we will try to encode the parity on a suitable subset, one for each x . Our encoding is implemented as follows. For each $x \in \{0, 1\}^n$, we define a segment $S_x \subset \{0, 1\}^I$ of enough size, roughly speaking, $20m/p_0$, which is polynomial in n . These segments are chosen so that the family $\{S_x\}_{x \in \{0, 1\}^n}$ is pair-wise disjoint. As in the proof of previous section, we would like to use the assignment of variables in S_x to encode the result of $\mathcal{M}(x; s(n))$. Here notice that the random restriction ρ has already assigned values to some of the variables in S_x . But since (i) $|S_x| = 20m/p_0$, and (ii) variables remain unassigned with probability p_0 , we can prove that with high probability, \dots segments S_x have at least $3m$ unassigned variables after the random restriction. We use these unassigned variables for encoding.

Thus, there exists a random restriction satisfying the following.

- (a) Each computation $\mathcal{M}(x; s(n))$ is reduced to a decision tree T_x of depth at most $2m$.
- (b) Each segment S_x has at least $3m$ unassigned variables, i.e., assigned $*$ by the restriction.

Fix ρ_0 to be one such restriction. Denote by Z_0 the set of variables in $\bigcup_{x \in \{0, 1\}^n} S_x$ that are assigned $*$ by ρ_0 , and rename variables so that $Z_0 = \{z_1, \dots, z_M\}$ and $Z - Z_0 = \{z_{M+1}, \dots, z_L\}$.

The restriction ρ_0 may assign $*$ to some variables in $Z - Z_0$, we now assign all such variables to 0. Then as explained above, the result of each computation of $\mathcal{M}(x; s(n))$ is expressed as a degree $2m$ polynomial $p_x(z_1, \dots, z_M)$ over the integers \mathbf{Z} . For each x , we try to equate $p_x(z_1, \dots, z_M)$ to the parity of S_x , i.e., $\bigoplus_{z_i \in S_x} z_i$. (Note that S_x contains variables not in $Z_0 = \{z_1, \dots, z_M\}$ whose values

are already fixed. By the term $\oplus_{z_i \in S_x} z_i$, we mean the parity of all variables in S_x (including such variables.) In other words, we wish to choose an assignment to z_1, \dots, z_M so that the following system of equations (*2) holds for $\{0, 1\}^n = \{x_1, \dots, x_N\}$, where $N = 2^n$.

$$(*2) \begin{cases} p_{x_1}(z_1, \dots, z_M) = \oplus_{z_j \in S_{x_1}} z_j, \\ \vdots \\ p_{x_N}(z_1, \dots, z_M) = \oplus_{z_j \in S_{x_N}} z_j. \end{cases}$$

Using a trick of exchanging 0,1 values by 1, -1 values, and reason about dimensions over a finite field \mathbf{Z}_3 , we can give an argument similar to the one in the previous section, and show that it is indeed possible to find such an assignment. Then the result follows.

Now we specify the parameters and the conditions explained above, and describe our proof precisely.

We focus on the simulation of some Σ_d^p machine $\mathcal{M}(x; s(n))$ on $N (= 2^n)$ inputs of length n for sufficiently large n . Let $m = O(n^k)$ be an integer bounding \mathcal{M} 's running time on length n inputs, and let $I = (1 + \delta)n$ and $L = 2^I$. We regard the computation of $\mathcal{M}(x; s(n))$ as a function over Boolean variables z_1, \dots, z_L , where each z_i is the boolean variable for a bit in $s(n)$. Furthermore, we may consider $\mathcal{M}(x; s(n))$ as a circuit C_x of depth $\leq d + 1$, size $\leq m2^m$, and bottom fan-in $\leq m$.

As explained above, we consider a random restriction to the variables z_1, \dots, z_L , with $p_0 = 1/(20m)^d$ being the probability $\Pr[z_i = *]$. For each $x \in \{0, 1\}^n$, the segment S_x is defined by $S_x = \{xu0^{\ell-n-n_0} : u \in \{0, 1\}^{n_0}\}$, where $n_0 = \lceil \log_2 20m/p_0 \rceil = \lceil (d + 1) \log_2 20m \rceil$. Clearly, any S_x and $S_{x'}$, for $x \neq x'$, are disjoint, and $|S_x|$ is of size larger than $20m/p_0$ but still polynomial in n .

We want some random restriction ρ , such that it satisfies the following two conditions.

- (a) For every $x \in \{0, 1\}^n$, the circuit C_x is reduced to a depth $t = 2m$ decision tree.
- (b) For every $x \in \{0, 1\}^n$, the segment S_x has at least $3m$ unassigned variables.

By using Lemma 1 and Chernoff's bound (see, e.g., Corollary A.1.14 of [AS00]), it is easy to show the following claim:

Under our choice of parameters, the probability that a random restriction ρ satisfies both (a) and (b) is not zero.

Hence, there exists some random restriction satisfying both (a) and (b).

Consider one of the restrictions ρ_0 satisfying both (a) and (b). We define $s(n)$ based on this ρ_0 ; that is, we will assign a bit in $s(n)$ to 0 or 1 according to ρ_0 . We will assign those variable assigned $*$ by ρ_0 later. Let Z_* be the set of variables assigned $*$ by ρ_0 . From condition (b) it follows that each S_x has at least $3m$ variables in Z_* . For each S_x , we pick lexicographically the first $3m$ such variables, and define Z_0 to be the set of those variables, over all x . Note that Z_0 has exactly $3mN$ variables because all S_x 's are disjoint. By renaming

for every ± 1 values of z_1, \dots, z_M , each p_x takes a ± 1 value, it follows that for every $+1, -1$ -assignment (a_1, \dots, a_M) to the z_i 's, there must be at least one x such that

$$p_x(a_1, \dots, a_M) = -\alpha_x \cdot \prod_{z_i \in Z_0 \cap S_x} a_i.$$

Thus, we have

$$\prod_{1 \leq i \leq N} \left[\alpha_{x_i} \prod_{z_j \in Z_0 \cap S_{x_i}} z_j + p_{x_i}(z_1, \dots, z_M) \right] = 0,$$

for all $+1, -1$ -assignments to z_1, \dots, z_M . Then it follows that the lefthand side expression is identical to 0 modulo the ideal $I = (z^2 - 1 : z \in Z_0)$. In other words, we have the identity

$$\prod_{1 \leq i \leq N} \prod_{z_j \in Z_0 \cap S_{x_i}} z_j = L(z_1, \dots, z_M)$$

in the ring $\mathbf{Z}_3[z_1, \dots, z_M]/I$, where L is a multilinear polynomial of degree at most $3m(N - 1) + 2m$. On the other hand, the lefthand side is multilinear and its degree is $3mN$, which is larger than $3m(N - 1) + 2m$. A contradiction.

This completes the proof of Theorem 3, and hence Theorem 2. With some more work one can show

Theorem 4. *Let $\delta > 0$ be a constant. Then for any polynomial p and any integer s such that $\ell(n) \geq 2^{(1+\delta)n}$, $\text{Mod}_p^{\text{PH}}/s = \text{P}/s$.*

5 Relations to the Conventional Relativized Results

Our nonuniform comparison is a restricted type relativization. Here we explain the position of our results and proofs in the existing relativized results.

First it should be noted that most relativized separation results are proved in a stringent way; that is, the proofs of such results can be modified easily for proving the same separation w.r.t. some advice function of some exponential (or super-polynomial) advice size. Most typically we can prove the following relation.

Proposition 1. *Let $\ell(n)$ be a function such that $\ell(n) \geq 2^{(1+\delta)n}$ for some constant $\delta > 0$. Then $\text{NP}/s \not\subseteq \text{P}/s$.*

Since our nonuniform notion is a generalization of the standard nonuniform model by Karp and Lipton, there are immediate implications for our nonuniform comparison from some of the results for the standard nonuniform model. For example, it has been known [Kan82] that $\text{PH} \not\subseteq \text{P}/p(n)$ for any fixed polynomial

$p(n)$. Since $\text{PH} \subseteq \text{PH}/s$ for any advice function s , the following fact is immediate from this result. This fact justifies the consideration of at least super-polynomial advice size for obtaining a nonuniform collapsing result for P and PH.

Proposition 2. *Let $\ell(n)$ be a super-polynomial function. Then $\text{PH}/s \subseteq \text{P}/s$ for any advice function s of advice size $\ell(n)$.*

While most relativized collapsing results are proved in a non stringent way, there are some relativized collapsing proofs in the literature that also yield nonuniform collapsing results in our context. For example, the following result is provable by a well-known technique; see, e.g., [Wil83]. (The proof is omitted here.)

Proposition 3. *Let $\ell(n) \geq 2^{(2+\delta)n}$ for some constant $\delta > 0$. Then $\text{NP}/s \subseteq (\text{P}/\text{poly})/s$ for any advice function s of advice size $\ell(n)$.*

By a similar proof technique, we can in fact prove $\text{NEXP} \subseteq \text{P}/\text{poly}$ in the standard relativization model [He86]. This is because for any given NEXP machine \mathcal{M} running in time $2^{p(n)}$, we can consider query strings of length $3p(n)$; since $2^{n+p(n)} < 2^{2p(n)}$, we still have enough room in $\{0, 1\}^{3p(n)}$ to encode the results of \mathcal{M} on all length n inputs. Some circuit of size $cp(n)$ for some sufficiently large $c > 0$ can retrieve this encoded information. On the other hand, this argument does not work in our context because the advice size cannot be bounded even exponentially. It should be also remarked here that a higher collapse is not immediate from a lower one in our context; for example, the relatively simple proof of $\text{NP}/s \subseteq (\text{P}/\text{poly})/s$ for some advice s of some exponential advice size bound does not give a proof of $\text{PH}/s' \subseteq (\text{P}/\text{poly})/s'$ for some s' of some exponential advice size bound. This latter result is indeed true, first proved by the authors using complicated arguments based on Nisan-Wigderson pseudorandom generators [CW03]. The results of the present paper give a simplified proof of a stronger result.

We believe that this model of random access to advice strings is an interesting model, which poses challenging problems. It is sufficiently different from the conventional relativization model for specific problems. Previous known proofs of relativized collapsing results do not, in general, imply the corresponding collapsing results in this model of random access to advice. Claims to the contrary should be first verified against the open problem of PSPACE vs. P.

References

[Ajt83] M. Ajtai, Σ_1^1 -formulae on finite structures, *Ann. Pure Applied Logic* 24, 1–48, 1983.
 [AS00] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., 2000.
 [BGS75] T. Baker, J. Gill, and R. Solovay, Relativizations of the P =? NP question, *SIAM J. Comput.* 4(4), 431–442, 1975.

- [BDG89] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I & II*, Springer, 1989 and 1990.
- [Cai86] J.-Y. Cai, With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy, in *Proc. 18th ACM Sympos. on Theory of Comput.* (STOC'89), 21–29, 1986. (The final version appeared in *J. Comp. Syst. Sci.* 38(1), 68–85, 1989.)
- [CW03] J.-Y. Cai and O. Watanabe, On proving circuit lower bounds against the polynomial-time hierarchy: positive and negative results, *Proc. 9th International Computing and Combinatorics Conference (COCOON'03)*, LNCS 2697, 202–211, 2003.
- [DK00] D. Du and K. Ko, *Theory of Computational Complexity*, John Wiley & Sons, Inc., 2000.
- [FSS81] M. Furst, J. Saxe, and M. Sipser, Parity, circuits, and the polynomial time hierarchy, in *Proc. 22nd IEEE Symposium on Foundations of Computer Science* (FOCS'81), IEEE, 260–270, 1981.
- [Hås86] J. Håstad, Almost optimal lower bounds for small depth circuits, in *Proc. 18th ACM Symposium on Theory of Computing* (STOC'86), ACM, 6–20, 1986.
- [He86] H. Heller, On relativized exponential and probabilistic complexity classes, *Information and Control* 71(3), 231–243, 1986.
- [Kan82] R. Kannan, Circuit-size lower bounds and non-reducibility to sparse sets, *Information and Control* 55, 40–56, 1982.
- [KL80] R. Karp and R. Lipton, Some connections between nonuniform and uniform complexity classes, in *Proc. 12th ACM Symposium on Theory of Computing* (STOC'80), ACM, 302–309, 1980. (An extended version appeared as: Turing machines that take advice, in *L'Enseignement Mathématique* (2nd series) 28, 191–209, 1982.)
- [Ko78] D. Kozen, Indexing of subrecursive classes, in *Proc. 10th ACM Symposium on Theory of Computing* (STOC'78), ACM, 287–295, 1978. (The final version appeared in *Theoretical Computer Science* 11, 277–301, 1980.)
- [Nis91] N. Nisan, Pseudorandom bits for constant depth circuits, *Combinatorica* 11(1), 63–70, 1991.
- [NW94] N. Nisan and A. Wigderson, Hardness vs randomness, *J. Comput. Syst. Sci.* 49, 149–167, 1994.
- [Raz87] A. Razborov, Lower bounds on the size of bounded depth networks over a complete basis with logical addition, *Mathematical Notes of the Academy of Sciences of the USSR* 41, 333–338, 1987.
- [Smo87] R. Smolensky, Algebraic methods in the theory of lower bounds for Boolean circuit complexity, in *Proc. 19th ACM Symposium on Theory of Computing* (STOC'87), ACM, 77–82, 1987.
- [Wil83] C.B. Wilson, Relativized circuit complexity, in *Proc. 24th IEEE Symposium on Foundations of Computer Science* (FOCS'83), IEEE, 329–334, 1983.
- [Yao85] A.C. Yao, Separating the polynomial-time hierarchy by oracles, in *Proc. 26th IEEE Symposium on Foundations of Computer Science* (FOCS'85), IEEE, 1–10, 1985.

Bounding the Payment of Approximate Truthful Mechanisms

Gruia Calinescu

Department of Computer Science, Illinois Institute of Technology,
Chicago, IL 60616
calinesc@iit.edu

Abstract. In a STACS 2003 paper, Talwar analyses the overpayment the VCG mechanism incurs for ensuring truthfulness in auction. Among other results, he studies k -Set Cover (given a universe U and a collection of sets S_1, S_2, \dots, S_q , each having a cost $c(S_i)$ and at most k elements of U , find a minimum cost subcollection, called cover, whose union equals U) and shows that the payment of the optimum cover OPT is at most $kc(OPT')$, where OPT' is the best cover disjoint from the optimum cover. For $k \geq 3$, k -Set Cover is known to be NP-Hard, and thus truthful mechanisms based on approximation algorithms are desirable. We show that the payment incurred by two approximation algorithms (including the Greedy algorithm) is bounded by $(k-1)c(OPT) + kc(OPT')$. The same approximation algorithms have payment bounded by $k(c(OPT) + c(OPT'))$ when applied to more general set systems, which include k -Polymatroid Cover, a problem with applications in Steiner Tree computations. If q is such that an element in a k -Set-Cover instance appears in at most q sets, we show that the payment of our algorithms is bounded by qk^2 times the payment of the optimum algorithm.

1 Introduction

There has been a surge of recent interest in the intersection area of economic sciences and computer science (see [21] for a survey). Inside this area is the field of *mechanism design*, which we describe below in the particular setting we are going to analyse in this paper. The seminal paper of Nisan and Ronen [20] introduced the computational issues of mechanism design and gives a general overview in a setting more general than ours. A reader with economics background will recognize the scenario in our simplified case. A reader with computer science background will find below all the definitions used in this paper, and is warned that more definitions and notations are required in the general setting.

We are given a set E of elements and a family \mathcal{F} of \dots subsets of E . We assume the set system (E, \mathcal{F}) is closed upwards, which means that for any $T \in \mathcal{F}$ and any superset $T' : T \subseteq T' \subseteq E$, we have $T' \in \mathcal{F}$. Each element $e \in E$ is controlled by a different economic agent, which we called the agent of e . The agent of e has a private (unknown to anyone else) cost $c(e) \geq 0$ for providing e

to a feasible subset. Finding feasible subsets is sometimes called *set selection* in the literature [3, 11] dealing with mechanisms.

A *mechanism* is a protocol which asks each agent to provide a bid $b(e)$, and then computes a feasible subset $M = M(b)$ and a payment function $p(e) \geq 0$. If $e \in M$, the agent of e must allow e to be used by M in exchange for the payment $p(e)$.

The agents are selfish and could misrepresent their cost in order to increase their payments. The field of *mechanism design* deals with the design of protocols which ensure the designer's goal are achieved by giving incentives to agents. The revelation principle [24, 18] states that in order to prove or disprove the existence of good mechanism for a given problem, one can consider only truthful revelation mechanism. Thus we only use direct revelation mechanisms, which are characterized by the computation of the feasible subset M and the payment function p .

In this context, the economics notion of maximizing social welfare is minimizing the cost of M . Minimizing the cost of M could be the goal of the designer of the mechanism. Another reasonable goal is minimizing the sum of payments. A mechanism is called *truthful* (also called *strategy-proof*) if every agent has interest to bid her cost (by setting $b(e) = c(e)$) regardless of the other bidders' bids.

Consider the simple case in which each element is a feasible subset, a situation which occurs frequently in real life. Say the government needs a task to be performed and invites sealed bids from agents (contractors). If the government chooses the lowest bidder and pays to this agent her bid, then the agents might bid higher than their cost with the goal of making a profit. The government can however use the *VCG mechanism* [26, 9, 12]: select the lowest bidder and pay her an amount equal to the second lowest bid. Assuming agent don't collude, it can be shown that in this case each agent has the interest to bid her cost. The VCG mechanism is truthful, and minimizes cost (maximizes social welfare). However the payment may be much larger than the cost.

The VCG mechanism can be applied to the set system setting as well [25, 3, 6], and selfish agent would bid their cost: for all e , we have $b(e) = c(e)$. The mechanism selects OPT , a feasible subset of minimum cost, and computes payments with a formula we present later.

The set systems we consider are instances of problems. For example, the elements can be the edges of a graph and feasible subsets consists of connected spanning subgraphs: the Minimum Spanning Tree problem. Of particular interest in routing protocols is the Shortest s - t Path problem [3, 11].

Let $p(I, \mathcal{A})$ be the payment of mechanism \mathcal{A} on instance I , that is the sum of the payments given by the mechanism to the agents. Talwar [25] analyses the *frugality ratio* of problems, which is defined as the maximum over instances I of $p(I, VCG)/c(OPT')$, where OPT' is the best feasible subset disjoint from OPT . [25] characterizes the problems with frugality ratio 1. For example, Minimum Spanning Tree has frugality ratio 1 [6, 25], and Shortest s - t Path has frugality ratio $\Theta(n)$ [3, 25], where n is the number of vertices in the graph.

One interesting problem analysed by Talwar [25] is the k -Set Cover problem: given a universe U and a collection of sets S_1, S_2, \dots, S_q , each having a cost $c(S_i)$ and at most k elements of U , find a minimum cost subcollection (called \dots) whose union equals U . k -Set Cover fits the general framework as follows: each set S_i is an element of E , and covers are the feasible subsets. [25] has proved that the frugality ratio of k -Set Cover is exactly k . Thus for k -Set Cover, VCG, besides optimizing the cost (social welfare), guarantees a bound on the payment.

For $k \geq 3$, k -Set Cover is known to be NP-Hard, and thus truthful mechanisms based on approximation algorithms are desirable. It is mentioned in [3, 1] that it is known that a mechanisms is truthful if it is \dots (defined later). For k -Set Cover, we analyze the monotone mechanism based on the GREEDY (G) approximation algorithms. We show that its payment is bounded by $(k-1)c(OPT) + kc(OPT')$. This bound is at most twice worse than the bound of the VCG mechanism, which must find the optimum solution. We present an instance of k -Set Cover where the payment of GREEDY is lower than the payment of OPT .

If q is such that an element in a k -Set-Cover instance appears in at most q sets, then we can prove the bound on the payments: $p(I, G) \leq k^2q \cdot p(I, VCG)$. We call (k, q) -Set Cover the Set Cover problem where each set has size at most k and every element appears in at most q sets. (k, q) -Set Cover is the only problem we are aware of where there is a nontrivial bound on the ratio of the payment of an approximation algorithm to the payment of the exact algorithm in our particular setting. Previous approximate truthful mechanisms [1, 2] either are for maximization problems and look at revenue (instead of payment) or look at a different setting in which costs are public and the agent have some other private data.

In the k -Polymatroid Cover problem, the set system is defined by a a sub-modular rank function $r : \mathcal{P}(E) \rightarrow \mathbb{N}$ with $r(e) \leq k$ for all $e \in E$: a feasible set T is a set satisfying $r(T) = r(E)$. The k -Polymatroid Cover problem has application for Steiner Tree computation [22, 5, 23, 28, 15, 14], and is a generalization of k -Set Cover. The k -Polymatroid problem is NP-Hard for $k \geq 3$, polynomial for $k = 1$ (when the problem becomes finding a minimum-cost basis in a matroid), while the complexity for $k = 2$ is unknown, but pseudopolynomial algorithms are known for linear polymatroids [7, 22]. Wolsey [27] has shown that GREEDY has approximation ratio H_k for k -Polymatroid Cover, where $H_k = 1 + \frac{1}{2} + \dots + \frac{1}{k}$ is the k th harmonic number. We show that $p(I, G) \leq k(c(OPT) + c(OPT'))$ and use techniques from [25] to show that $p(I, VCG) \leq kc(OPT')$.

A second algorithm applies any set system with an oracle deciding whether a set is feasible or not. Space limitations prevent us from discussing this algorithm, and we just mention that we can prove exactly the same bounds on its payment as we do for GREEDY when the set system is given by k -Set Cover, (k, q) -Set Cover, or k -Polymatroid Cover.

The paper is organized as follows. Section 2 gives the definition of monotone mechanisms with some known associated properties, definitions related to k -Polymatroids, and the definitions of the GREEDY algorithm, with some as-

sociated properties. Section 3 compares the payment of GREEDY to $c(OPT) + c(OPT')$. Section 4 compares the payment of the approximation algorithm directly to the payment of the optimum algorithm.

2 Preliminaries

We start by formally defining truthfulness. Let $M(b)$ be the feasible subset computed by the mechanism given bid vector $b = b(e) \mid e \in E$, where E is the set of elements. For agent e and bid vector b , we define $profit_e(b) = p(e) - c(e)$, if $e \in M(b)$ and $profit_e(b) = 0$ otherwise. We assume that each agent's goal is to maximize her profit. The algorithm which computes $M(b)$ is known to all before the bidding, and the payments $p(e)$ are known to all after the bidding. We require that our mechanisms satisfy the *incentive compatibility* condition, which states that the profit of an agent cannot be negative (or the agent might decide not to participate).

Given a mechanisms as described above, let b_{-e} denote the vector of all bids except the one made by the agent of e , so we can write $b = (b_{-e}, b(e))$. We say that *truth-telling* is a *dominant strategy* for the agent of e if, regardless of b_{-e} , bidding $b(e) = c(e)$ maximizes $profit_e(b)$. So, even if the agent of e knew the bids of the other agents ahead of time, the best she could do is bid the truth cost. If truth-telling is a dominant strategy for every agent we call the mechanism *truthful*.

2.1 Monotone Mechanisms

Let $b(e)$ be the bid values of the agents, and let $M(b)$ be the subset picked by the mechanism, which depends on the bids b . The mechanism is *monotone* if for every element $e \in M(b)$ a lower bid $\bar{b}(e) < b(e)$ only by the agent of e (that is, $\bar{b}(f) = b(f)$ for $f \in E \setminus \{e\}$), also result in $e \in M(\bar{b})$. In this case there is a threshold $t(e) = t(e, b_{-e})$ for every $e \in E$ which depends on the algorithms which picks M and b_{-e} , such that:

- If $b(e) < t(e, b_{-e})$, then $e \in M(b)$
- If $b(e) > t(e, b_{-e})$, then $e \notin M(b)$

One defines

$$p(e, b) = \begin{cases} 0 & \text{if } e \notin M(b) \\ t(e, b_{-e}) & \text{if } e \in M(b) \end{cases}$$

If $M(b)$ is an optimum solution, then this mechanism is monotone, and in fact coincides with the VCG mechanism, which is defined as follows. Let $b[e \rightarrow x]$ denote the cost function \bar{b} which is equal to b everywhere except that $\bar{b}(e) = x$. Let $opt(b) = b(OPT)$, the total bid-value of the optimum solution for bids b .

Then:

$$p(e, b) = \begin{cases} 0 & \text{if } e \notin OPT(b) \\ opt([b \rightarrow \infty]) - opt([b \rightarrow 0]) & \text{if } e \in OPT(b) \end{cases}$$

For the sake of completeness we include a proof of the folklore [3] result:

Theorem 1.

Proof. First we note that since $p(e, b) \geq b(e)$ whenever $e \in M$, and $p(e, b) = 0$ whenever $e \notin M(b)$ (this is the case when no services or money is traded), the agent has an interest to participate in the mechanism. Why would an agent bid other than her cost? If the agent of e bids $b(e) < c(e)$, then three outcomes are possible:

- $e \notin M(b)$. The agent of e does not trade services or money, the same outcome as in the case when $b(e) = c(e)$ (the monotonicity of the mechanism implies that $e \notin M$ in when $b(e) = c(e)$ as well)
- $e \in M(b)$ and $c(e) > t(e, b_{-e})$. In this case the agent of e is losing money (as $p(e, b) = t(e, b_{-e})$), while bidding $b(e) = c(e)$ would result in no trade or a trade with payment being equal to the cost.
- $e \in M(b)$ and $c(e) \leq t(e, b_{-e})$. In this case the agent receives the same payment as in the case when she bids $b(e) = c(e)$

On the other side, if the agent of e bids $b(e) > c(e)$, then again three outcomes are possible:

- $e \in M(b)$. As $t(e, b_{-e}) \geq b(e)$, the payment received by the agent of e is the same as in the case when it bids $b(e) = c(e)$ (the monotonicity of the mechanism implies that $e \in M$ in this case as well)
- $e \notin M(b)$ and $c(e) < t(e, b_{-e})$. In this case the agent does not trade, while bidding $b(e) = c(e)$ would have resulted in a payment exceeding the cost
- $e \notin M(b)$ and $c(e) \geq t(e, b_{-e})$. In this case the agent does not trade. Bidding $b(e) = c(e)$ would have resulted also in a no-trade, or in a trade with payment being equal to the cost.

In conclusion, in no case can the agent benefit from not bidding her cost. ■

Xiang-Yang Li and Weizhao Wang [16] have implicitly proved that the GREEDY algorithm when applied to Set Cover is monotone.

2.2 Polymatroids and Approximation Algorithms

A rank function $r : \mathcal{P}(E) \rightarrow \mathbb{N}$ is called submodular if for any two sets $A, B \subseteq E$

$$r(A \cap B) + r(A \cup B) \leq r(A) + r(B)$$

A k -polymatroid is given by a submodular rank function satisfying $r(\{e\}) \leq k$ for all $e \in E$. The k -Polymatroid Cover problems asks for a minimum cost subset of E with rank equal to $r(E)$. k -Polymatroid Cover is a generalization of k -Set Cover by setting for a collection of sets A the rank $r(A) = |\cup_{S \in A} S|$. The GREEDY algorithm for Set Cover analysed by Chvatal [8] can be used for k -Polymatroid Cover [27] as follows: let A be the set of elements picked so far; initially $A = \emptyset$. If $r(A) < r(E)$, find $g \in E$ such that $\frac{r(A \cup \{g\}) - r(A)}{c(g)}$ is maximized, and set $A \leftarrow A \cup \{g\}$. We generalize and simplify [16] to obtain:

Theorem 2. GREEDY k

Proof. Assume we decrease the bid of e from $b(e)$ to $\bar{b}(e)$, while for the other elements $f \in E$ we keep $\bar{b}(f) = b(f)$. The GREEDY algorithm proceeds as before the decrease and picks e , unless e is picked sooner. ■

Since we only use monotone algorithms in this paper, and truthful mechanisms based on them, from now on we do not mention the bids, as we assume the bids equal the costs.

3 The GREEDY Algorithm

In this section we compare the payment of the GREEDY algorithm to $c(OPT')$. We start with k -Set Cover, where some arguments of Talwar [25] help.

Theorem 3. $p(G) \leq (k - 1)c(OPT) + kc(OPT')$

Proof. Let $T = S_1, S_2, \dots, S_n$ be the collection of sets and let B_1, B_2, \dots, B_m be the sets picked by GREEDY in this order. Let Q_i be the subset of B_i with the elements first covered by B_i (that is, not covered by B_1, B_2, \dots, B_{i-1}), and note that for $1 \leq i < j \leq m$, we have $Q_i \cap Q_j = \emptyset$.

Consider a set B_i and we analyze $p(B_i)$. We increase $c(B_i)$ until GREEDY does not pick B_i . We reach the following situation: GREEDY picks B_1, B_2, \dots, B_{i-1} , then it might pick a several sets H_j , and then it picks B_i . However, a further increase in $c(B_i)$ results in B_i not being picked. Let \bar{c} be this new cost function which differs from c only for B_i , and note that $p(B_i) = \bar{c}(B_i)$. Define \bar{Q}_i to be the elements first covered by B_i in the cover given by GREEDY when run with \bar{c} , and note that $\bar{Q}_i \subseteq Q_i$.

Assume first that $B_i \notin OPT'$. We know \bar{Q}_i is covered by sets R_1, R_2, \dots, R_a of OPT' , and note that GREEDY did not pick any of R_j before B_i when run on cost \bar{c} . By the way GREEDY operates, for $j = 1, 2, \dots, a$:

$$c(R_j) \geq \frac{|R_j \cap \bar{Q}_i|}{|\bar{Q}_i|} \bar{c}(B_i). \tag{1}$$

Summing up over j , we obtain that $\bar{c}(B_i) \leq \sum_{j=1}^a c(R_j)$, and we say that B_i charges each R_j . Since $|R_j| \leq k$ and the sets \bar{Q}_i are disjoint, it follows that a set R in OPT' can be charged at most k times. We have to do more work since when $B_i \in OPT'$, we cannot always find sets in OPT' to be charged as above by B_i .

Now assume that $B_i \in OPT'$. Using the argument as above, we find several $O_j \in OPT$ such that B_i charges O_j , and O_j is charged in total at most k times. At this moment we have proven $p(G) \leq k(c(OPT') + c(OPT))$.

If O_j from OPT is charged exactly k times, then it is charged by k sets from OPT' . Each of these k sets of OPT' is using a different element to charge O_j

and therefore they together cover O_j . Since O_j can be replaced by these k sets of OPT' , their total cost is at least $c(O_j)$. We move the charges as follows: each of the k sets B_i of OPT' which charges O_j is asked to charge only $c(O_j) - c(B_i)$; by the argument above this reduces the charge of O_j to at most $(k - 1)c(O_j)$. B_i is charging $c(B_i)$ to itself. Now B_i can be charged by other sets of G , but it can be charged at most $k - 1$ times since it cannot be charged through the element it uses to charge O_j , as that element is in \bar{Q}_i and not in any other \bar{Q}_r , with B_r in G . Thus B_i is charged in total at most k times.

With the modified charges, each set in OPT is charged at most $k - 1$ times and each set of OPT' is charged at most k times. Thus we have $p(G) \leq (k - 1)c(OPT) + kc(OPT')$. ■

We do not have a tight example for the bound in Theorem 3 and suspect the bound is not tight. We include an example (see also Figure 1) taken from [25] which shows $p(OPT) = k \cdot c(OPT')$ and for which $p(G) = k \cdot c(OPT')$ too. There are $k + 1$ sets S_0, S_1, \dots, S_k of costs $c(S_0) = 1$ and $c(S_i) = 0$ for $i = 1, 2, \dots, k$. All the k elements e_1, e_2, \dots, e_k are included in S_0 , while $S_i = \{e_i\}$. GREEDY (and also OPT) picks the 0-cost sets and one can check it pays 1 for each of them.

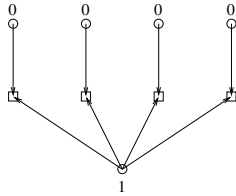


Fig. 1. An example where the payment $p(OPT) = p(G) = k \cdot c(OPT')$ for $k = 4$. The sets are represented by circles and the elements by squares. GREEDY (and also OPT) picks the sets of cost 0 and pays 1 for each of them, while $c(OPT') = 1$

For the more general k -Polymatroid, we obtain a slightly weaker bound.

Theorem 4. For any k -Polymatroid (E, \mathcal{F}) with $r(E) = k$ and any $(G, \mathcal{G}) \in \text{OPT}$, we have $p(G) \leq k \cdot c(OPT) + c(OPT')$.

Proof. Some of the arguments below are taken from the analysis of the GREEDY algorithm given in [4]. Helgason [13], McDiamid [19], and Lovasz [17] have shown that each polymatroid (E, \mathcal{F}) can be obtained from a matroid M with rank function \bar{r} by the following construction: An element $h \in E$ corresponds to a set $M_h \in M$ such that for all $X \subseteq E$ we have $r(X) = \bar{r}(\cup_{h \in X} M_h)$.

Let $n := r(E) = \bar{r}(M)$ and let g_1, g_2, \dots, g_m be, in this order, the elements picked by GREEDY. Let $B_i = M_{g_i}$, as defined above. Let $S_0 := A_0 := \emptyset$, and for $i = 1, 2, \dots, m$, let $A_i \subseteq M$ be defined by $A_i = \cup_{j=1}^i B_j$. For $i = 1, 2, \dots, m$, let $Q_i \subseteq B_i$ be such that $S_i := \cup_{j=1}^i Q_j$ is a basis for A_i .

Consider an element g_i picked by GREEDY and we analyze $p(g_i)$. We increase $c(g_i)$ until GREEDY does not pick g_i . We reach the following situation: GREEDY picks g_1, g_2, \dots, g_{i-1} , then it might pick a set of elements H , and then it picks g_i . However, a further increase in $c(g_i)$ results in g_i not being picked. Let \bar{c} be this new cost function which differs from c only for g_i . Define $Z_i \subseteq \cup_{h \in H} M_h$ such that $S_{i-1} \cup Z_i$ is a basis for $A_{i-1} \cup (\cup_{h \in H} M_h)$. Now define \bar{Q}_i such that $\bar{Q}_i \subseteq Q_i$ and $S_{i-1} \cup Z_i \cup \bar{Q}_i$ is a basis in $A_{i-1} \cup (\cup_{h \in H} M_h) \cup Q_i$, which means $S_{i-1} \cup Z_i \cup \bar{Q}_i$ is also a basis in $A_{i-1} \cup (\cup_{h \in H} M_h) \cup B_i$ since $S_{i-1} \cup Q_i$ spans $A_{i-1} \cup B_i$.

Let $T \subseteq \cup_{h \in OPT'} M_h$ be another basis of M . For every $f \in T$ choose a $\psi(f) \in OPT'$ such that $f \in M_{\psi(f)}$. For all $h \in OPT'$, the set $\psi^{-1}(h)$ is independent and therefore:

$$|\psi^{-1}(h)| \leq r(h) \leq k \tag{2}$$

In order to analyse the payment of GREEDY, we need to select from T disjoint sets \bar{X}_i , with $i = 1, 2, \dots, m$, such that $|\bar{X}_i| = |\bar{Q}_i|$ and each $A_{i-1} \cup Z_i \cup \bar{X}_i$ is independent. We ensure disjointness by constructing a partition of T into sets X_i , with $i = 1, 2, \dots, m$ such that $\bar{X}_i \subseteq X_i$. Borrowing again from [4], we construct \bar{X}_i and X_i one by one, but in a more involved procedure described below. Let $T_0 := T$ and $T_i := T \setminus (\cup_{j=1}^i X_j)$. We impose the invariant that for all $i = 0, 1, \dots, m$, $S_i \cup T_i$ forms a base in M .

We process $i = 1, 2, \dots, m$ as follows. We complete $S_{i-1} \cup Z_i \cup \bar{Q}_i$ to a base using a set $W_i \subseteq T_{i-1}$; such a completion exists since $S_{i-1} \cup T_{i-1}$ forms a base. We contract $S_{i-1} \cup W_i$, obtaining a new matroid M_i , in which we have two bases: $Z_i \cup \bar{Q}_i$ and $T_{i-1} \setminus W_i$. Using Exercise 8.48 from [10] or its generalization in Lemma 1 below, $T_{i-1} \setminus W_i$ can be partition into two sets, one of them we call \bar{X}_i , such that $(T_{i-1} \setminus W_i \setminus \bar{X}_i) \cup \bar{Q}_i$ and $Z_i \cup \bar{X}_i$ are both bases in the matroid M_i . Thus $S_{i-1} \cup Z_i \cup \bar{X}_i$ is independent and $S_{i-1} \cup \bar{Q}_i \cup (T_{i-1} \setminus \bar{X}_i)$ is a base in M .

Now we can use the procedure described in [4] to get X_i such that $S_i \cup (T_{i-1} \setminus X_i)$ is a base. If we denote $F_i := Q_i \setminus \bar{Q}_i$, we have $S_i = S_{i-1} \cup Q_i = S_{i-1} \cup \bar{Q}_i \cup F_i$. If $F_i = \emptyset$, we set $X_i = \bar{X}_i$ and we can proceed to $i+1$. Else, let f_1, \dots, f_r be the elements of F_i . Starting with $j = 1$, we find elements $t_j \in T_{i-1} \setminus \bar{X}_i \setminus \{t_1, t_2, \dots, t_{j-1}\}$ such that $S_{i-1} \cup \bar{Q}_i \cup \{f_1, f_2, \dots, f_j\} \cup (T_{i-1} \setminus \bar{X}_i \setminus \{t_1, t_2, \dots, t_j\})$ is a base. The element t_j is found as follows: If $f_j \in (T_{i-1} \setminus \bar{X}_i \setminus \{t_1, t_2, \dots, t_{j-1}\})$, then $t_j = f_j$. Else, adding f_j to the base $S_{i-1} \cup \bar{Q}_i \cup \{f_1, f_2, \dots, f_{j-1}\} \cup (T_{i-1} \setminus \bar{X}_i \setminus \{t_1, t_2, \dots, t_{j-1}\})$ creates a circuit C . As f_j is not spanned by $S_{i-1} \cup \bar{Q}_i \cup \{f_1, f_2, \dots, f_{j-1}\}$, we can find $t_j \in C \setminus (S_{i-1} \cup \bar{Q}_i \cup \{f_1, f_2, \dots, f_{j-1}\})$. It follows that $S_{i-1} \cup \bar{Q}_i \cup \{f_1, f_2, \dots, f_{j-1}\} \cup (T_{i-1} \setminus \bar{X}_i \setminus \{t_1, t_2, \dots, t_{j-1}\}) \setminus \{t_j\} \cup \{f_j\}$ is also a base. Then we set $X_i = \bar{X}_i \cup \{t_1, t_2, \dots, t_j\}$ and we note that indeed $S_i \cup T_i$ forms a base thus maintaining the invariant.

Thus we selected from T disjoint sets \bar{X}_i , with $i = 1, 2, \dots, m$, such that $|\bar{X}_i| = |\bar{Q}_i|$ and each $A_{i-1} \cup Z_i \cup \bar{X}_i$ is independent.

Assume first that $g_i \notin OPT'$. We know B_i increases the rank of $S_{i-1} \cup Z_i$ by $|\bar{Q}_i|$. \bar{X}_i also increases the rank of $S_{i-1} \cup Z_i$ by $|\bar{X}_i| = |\bar{Q}_i|$. Let h_1, h_2, \dots, h_a be elements of OPT' such that $\psi(h_i) \cap \bar{X}_i \neq \emptyset$, and note that GREEDY did not

pick any of h_j before g_i when run on cost \bar{c} . Define $r_j = r(S_{i-1} \cup Z_i \cup M_{h_j}) - r(S_{i-1} \cup Z_i)$. By the way GREEDY operates, for $j = 1, 2, \dots, a$:

$$c(h_j) \geq \frac{r_j}{|\bar{Q}_i|} \bar{c}(g_i). \tag{3}$$

Summing up over j and using the fact (obtained from submodularity) that $\sum_{j=1}^a r_j \geq r(S_{i-1} \cup Z_i \cup \bar{X}_i) - r(S_{i-1} \cup Z_i) = |\bar{X}_i| = |\bar{Q}_i|$, we obtain that $\bar{c}(g_i) \leq \sum_{j=1}^a c(h_j)$, and we say that g_i charges each h_j . From Equation 2 and the fact that the sets \bar{X}_i are disjoint we obtain that each element of OPT' can be charged at most k times.

Now assume that $g_i \in OPT'$. Using the same argument, but with the basis T derived from OPT rather than OPT' , we charge the payment of g_i to the cost of one or more elements of OPT , and such that each element of OPT can be charged at most k times. At this moment we have proven $p(G) \leq k(c(OPT') + c(OPT))$. ■

The following property of matroids generalizes Exercise 8.48 of [10]. Due to lack of space we omit the proof of the lemma and the following theorem, which uses Talwar’s technique and the lemma.

Lemma 1. Let M and N be matroids on A, B and M and N be matroids on A, B . Let X_1, X_2, \dots, X_q and Y_1, Y_2, \dots, Y_q be subsets of $A \cup B$ such that $X_i \cap Y_i = \emptyset$ and $X_i \cap Y_j = \emptyset$ for $i \neq j$. Let $G = (A \cup B, E)$ where $E = \{X_i \cup Y_i \mid i = 1, 2, \dots, q\}$.

Theorem 5. Let (E, \mathcal{F}) be a matroid with k independent sets S_1, \dots, S_k and c be a cost function on E . Let OPT and OPT' be optimal solutions to the matroid problem on (E, \mathcal{F}) with cost function c . Then $p(OPT) \leq kc(OPT')$.

We include a k -Set-Cover example that show that the payment of GREEDY can be lower than the payment of OPT . We use $k = 2$; note that this problem, also known as Edge Cover, has polynomial-time algorithms. An illustration of the example is given in Figure 2. OPT consists of three sets: $\{1, 2\}$, $\{3, 4\}$, and $\{5, 6\}$ each of cost $1 + \epsilon$. OPT' , which is also G , consist of four sets each of cost 1: $\{1, 3\}$, $\{2, 3\}$, $\{4, 5\}$, and $\{4, 6\}$. One can check that $p(OPT) = 3(2 - 2\epsilon)$ while $p(G) = 4(1 + \epsilon)$.

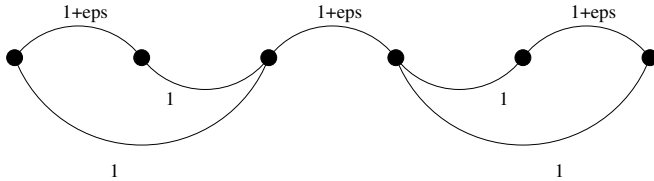


Fig. 2. An example where the payment of OPT is larger than the payment of the approximation algorithm GREEDY. The sets are the edges of this graph

4 Bounds in Terms of $p(OPT)$

In this section we analyse the payment of the approximation algorithms versus the payment of the optimum algorithm. For k -Set-Cover, the payment of GREEDY can be much higher than the payment of OPT , as shown by the following example, also illustrated in Figure 3. We have $1 + q(k - 1)$ elements $d_0, d_1, \dots, d_{q(k-1)}$. Set $S_0 = \{d_0\}$ and $c(S_0) = 1$. For $i = 1, 2, \dots, q(k - 1)$, set $S_i = \{d_i\}$ and $c(S_i) = 0$. For $j = 1, 2, \dots, q$, the set $B_j = \{d_0\} \cup \{d_{(j-1)(k-1)+1}, d_{(j-1)(k-1)+2}, \dots, d_{j(k-1)}\}$, each of cost $1 + \epsilon$. One can check $p(OPT) = 1 + \epsilon(1 + q(k - 1))$. GREEDY also finds the optimum solution, and one can check that for each $i = 1, 2, \dots, q(k - 1)$, GREEDY pays $\frac{1+\epsilon}{2}$. Thus $p(G) \geq (\frac{1}{2} - \epsilon)(k - 1)qp(OPT)$ is possible.

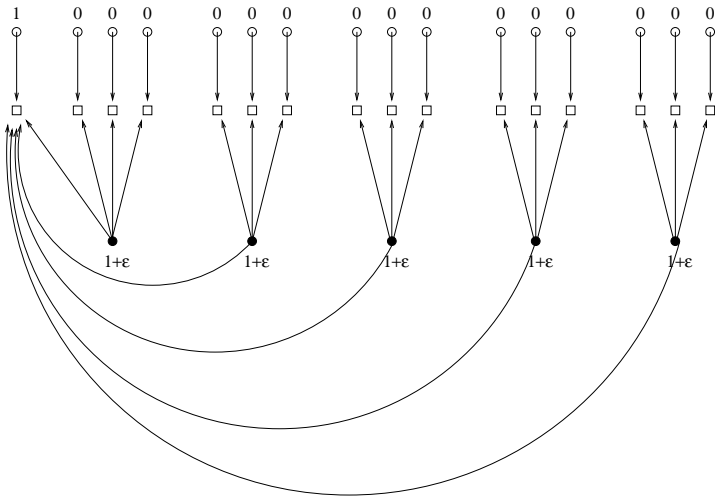


Fig. 3. An example where $p(G)$ is $\Omega(kq)$ higher than $p(OPT)$, for $k = 4$ and $q = 5$

We can bound the payments of the approximation algorithm as follows:

Theorem 6. $k \geq 2$ and $q \geq 2$ implies $p(G) \leq qk^2 \cdot p(OPT)$

Proof. We assume without loss of generality that $p(OPT)$ is finite, which implies that for every set S in the optimum cover, there is a cover without S .

We use arguments and notation from Theorem 3 repeatedly. Let B_1, B_2, \dots, B_m be the sets picked by GREEDY in this order. Let Q_i be the subset of B_i with the elements first covered by B_i (that is, not covered by B_1, B_2, \dots, B_{j-1}), and note that for $1 \leq i < j \leq m$, we have $Q_i \cap Q_j = \emptyset$.

Consider a set B_i and we analyze $p(B_i)$. We increase $c(B_i)$ until GREEDY does not pick B_i . We reach the following situation: GREEDY picks B_1, B_2, \dots, B_{i-1} ,

then it might pick a several sets H_j , and then it picks B_i . However, a further increase in $c(B_i)$ results in B_i not being picked. Let \bar{c} be this new cost function which differs from c only for B_i , and note that $p(B_i) = \bar{c}(B_i)$. Define \bar{Q}_i to be the elements first covered by B_i in the cover given by GREEDY when run with \bar{c} , and note that $\bar{Q}_i \subseteq Q_i$.

Define Z_i to be the set of elements not covered by $OPT \setminus \{B_i\}$ (so $Z_i = \emptyset$ if $B_i \notin OPT$). If $B_i \in OPT$, define M_i and N_i to be collection of sets such that $M_i \cup N_i$ is a minimum cost cover which does not use B_i and with M_i covering $Z_i \cap \bar{Q}_i$. For convenience define $M_i = \emptyset$ if $B_i \notin OPT$. Define U_i to be some collection of sets from OPT which cover $\bar{Q}_i \setminus Z_i$.

The arguments from Theorem 3 give:

$$p(G, B_i) \leq \sum_{R_j \in M_i} c(R_j) + \sum_{R_j \in U_i} c(R_j). \quad (4)$$

Also, each $R_j \in OPT$ appears for at most k indexes i in sets U_i as the sets \bar{Q}_i are disjoint and R_j has at most k elements. Note that, as in the VCG mechanism,

$$p(OPT, B_i) = c(M_i) + c(N_i) - c(OPT) - c(B_i), \quad (5)$$

where $c(M_i) = \sum_{R_j \in M_i} c(R_j)$ and $c(N_i) = \sum_{R_j \in N_i} c(R_j)$. Let V_i be the collection of sets of $OPT \setminus \{B_i\}$ which cover $\cup_{R_j \in M_i} R_j \setminus B_i$.

Consider the set of elements F which are not covered by any set of $V_i \cup \{B_i\}$. We have two covers for F : $OPT \setminus V_i \setminus \{B_i\}$ and N_i . The optimality of OPT implies that $c(N_i) \geq c(OPT) - c(V_i) - c(B_i)$. Plugging this into Equation 5 gives:

$$c(M_i) = p(OPT, B_i) - c(N_i) + c(OPT) + c(B_i) \leq p(OPT, B_i) + 2c(B_i) + c(V_i). \quad (6)$$

We say that B_i charges S , for set $S \in OPT$, if $S \in V_i$. We count how many times can S be charged. For B_i to charge S , there must be a set R and an element $e \in \bar{Q}_i \cap R$ such that $R \cap S \neq \emptyset$. There are at most $k(q-1)$ such sets R , and not counting the elements of S , the cover at most $k(k-1)(q-1)$ elements. Noting that the sets \bar{Q}_i are disjoint and therefore no two B_i can charge S through the same e , we obtain that S can be charged at most $k(k-1)(q-1)$ times.

Now Equations 4 and 6 give

$$p(G, B_i) \leq c(U_i) + p(OPT, B_i) + 2c(B_i) + c(V_i). \quad (7)$$

Summing over i and taking into account the number of charges, we obtain

$$p(G) \leq kc(OPT) + p(OPT) + 2c(OPT) + k(k-1)(q-1)c(OPT). \quad (8)$$

As $c(OPT) \leq p(OPT)$ and $k, q \geq 2$, we conclude that $p(G) \leq qk^2p(OPT)$. ■

Acknowledgments

We are grateful to Xiang-Yang Li and Weizhao Wang for providing us with an early draft of [16], thus inspiring this work. We also thank Bill Cunningham for discussions on matroids.

References

1. A. Archer, C. Papadimitriou, K. Talwar and E. Tardos, "An Approximate Truthful Mechanism for Combinatorial Auctions with Single Parameter Agents," *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 205–214, 2003.
2. A. Archer and E. Tardos. "Truthful Mechanisms for One-Parameter Agents," *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, pp. 482–491, 2001.
3. A. Archer and E. Tardos. "Frugal path mechanisms," *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 991–999, 2002.
4. G. Baudis, C. Gropl, S. Hougardy, T. Nierhoff, and H. J. Promel. "Approximating minimum spanning sets in hypergraphs and polymatroids," *ICALP*, 2000.
5. P. Berman and V. Ramaiyer. "Improved Approximations for the Steiner Tree Problem," *J. Algorithms*, vol. 17, pp. 381–408, 1994.
6. S. Bikhchandani, S. de Vries, J. Schummer, and R. Vohra. "Linear programming and Vickrey auctions," *IMA Volumes in Mathematics and its Applications, Mathematics of the Internet: E-Auctions and Markets* vol. 127, pp. 75–116, 2001.
7. P.M. Camerini, G. Galbiati, and F. Maffioli. "Random pseudo-polynomial algorithms for exact matroid problems," *J. Algorithms*, vol. 13, pp. 258–273, 1992.
8. V. Chvatal. "A greedy heuristic for the set covering problem," *Mathematics of Operation Research*, vol. 4, pp. 233–235, 1979.
9. E. H. Clarke. "Multipart pricing of public goods," *Public Choice*, vol. 8, pp. 17–33, 1971.
10. W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience, 1998.
11. E. Elkind, A. Sahai, and K. Steiglitz. "Frugality in path auctions," *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 701–709, 2004.
12. T. Groves. "Incentive in teams," *Econometrica*, vol. 41(4), pp. 617–631, 1973.
13. T. Helgason. "Aspects of the theory of hypermatroids," *Hypergraph Seminar: Ohio State University*, pp. 191–213, 1974.
14. S. Hougardy and H. J. Promel. "A 1.598 approximation algorithm for the Steiner problem in graphs," *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 448–453, 1999.
15. M. Karpinski and A. Zelikovsky. "New Approximation Algorithms for the Steiner Tree Problems," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 2(030), 1995.
16. X.-Y. Li and W. Wang. "Efficient Strategyproof Multicast in Selfish Networks," *International Workshop on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*, 2004.
17. L. Lovasz. "Flats in matroids and geometric graphs," *Sixth British combinatorial conference*, pp. 45–86, 1977.

18. A. Mas-Colle, W. Whinston, and J. Green. *Microeconomic Theory* Oxford university press, 1995.
19. C. McDiarmid. "Rado's theorem for polymatroids," *Math. Proc. Cambridge Philos. Soc.*, vol. 78, pp. 263–281, 1975.
20. N. Nisan and A. Ronen. "Algorithmic mechanism design," *Proceedings of the thirty-first annual ACM symposium on Theory of computing* , pp. 129–140, 1999.
21. C. Papadimitriou. "Algorithms, games, and the Internet," *Proceedings of the thirty-third annual ACM symposium on Theory of computing* , pp. 749–753, 2001.
22. H.J. Promel and A. Steger. "A new approximation algorithm for the Steiner tree problem with performance ratio $5/3$," *J. of Algorithms*, vol. 36, pp. 89–101, 2000.
23. G. Robins and A. Zelikovsky. "Improved Steiner Tree Approximation in Graphs," *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 770–779, 2000.
24. A. Ronen. "Algorithms For Rational Agents," *SOFSEM*, 2000.
25. K. Talwar. "The price of truth: Frugality in truthful mechanisms," *20th Annual Symposium on Theoretical Aspects of Computer Science* , pp. 608–619, 2003.
26. W. Vickrey. "Counterspeculation, auctions and competitive sealed tenders," *Journal of Finance*, vol. 16, pp. 8–37, 1961.
27. L.A. Wolsey. "Analysis of the greedy algorithm for the submodular set covering problem," *Combinatorica*, vol. 2, pp. 385–392, 1982.
28. A. Zelikovsky. "An $11/6$ -Approximation Algorithm for the Network Steiner Tree Problem," *Algorithmica*, vol. 9, pp. 463–470, 1993.

The Polymatroid Steiner Problems

Gruia Calinescu¹ and Alexander Zelikovsky²

¹ Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616

calinesc@iit.edu

² Department of Computer Science, Georgia State University, Atlanta, GA 30303

alexz@cs.gsu.edu

Abstract. The Steiner tree problem asks for a minimum cost tree spanning a given set of terminals $S \subseteq V$ in a weighted graph $G = (V, E, c)$, $c : E \rightarrow R^+$. In this paper we consider a generalization of the Steiner tree problem, so called Polymatroid Steiner Problem, in which a polymatroid $P = P(V)$ is defined on V and the Steiner tree is required to span at least one base of P (in particular, there may be a single base $S \subseteq V$). This formulation is motivated by the following application in sensor networks – given a set of sensors $S = \{s_1, \dots, s_k\}$, each sensor s_i can choose to monitor only a single target from a subset of targets X_i , find minimum cost tree spanning a set of sensors capable of monitoring the set of all targets $X = X_1 \cup \dots \cup X_k$. The Polymatroid Steiner Problem generalizes many known Steiner tree problem formulations including the group and covering Steiner tree problems. We show that this problem can be solved with the polylogarithmic approximation ratio by a generalization of the combinatorial algorithm of Chekuri et. al. [7].

We also define the Polymatroid directed Steiner problem which asks for a minimum cost arborescence connecting a given root to a base of a polymatroid P defined on the terminal set S . We show that this problem can be approximately solved by algorithms generalizing methods of Charikar et al [6].

Keywords: Wireless sensor networks, Steiner trees, polymatroid, approximation algorithms.

1 Introduction

This paper is motivated by the following lifetime problem in energy-constrained sensor networks. Let S be a set of (stationary) sensors which can be employed for monitoring a set X of (possibly moving) targets. Each sensor $s_i \in S$ can monitor at most one target chosen from $X_i \subseteq X$, a subset of targets visible to s_i . All targets are supposed to be simultaneously monitored by activated sensors which should continuously transmit collected data to the base possibly using multi-hop connections through other sensors, i.e., the activated sensors and the base should be connected with a Steiner tree. A *schedule* is a set of pairs (T, t) , where T a Steiner tree connecting sensors capable of monitoring all targets and t is time during which T is used. A simple energy model assumes that all sensors transmit with a single unit power and the Steiner tree is derived from the unit-disk graph. Then the energy consumption of each sensor is proportional to the time t during which it is used.

Target-Monitoring Sensor Network Lifetime Problem. Find a schedule of the maximum total time span such that each sensor $s_i \in S$ does not exceed given initial energy supply b_i .

Previously, several versions of the communication adhoc network lifetime problems as well as sensor network lifetime problem have been explored in [4] and [3], respectively. A provably good approach to the lifetime problems consists of the following steps:

- (i) formulating the lifetime problem as a packing linear program,
- (ii) approximately solving the dual covering linear program,
- (iii) applying the primal-dual algorithm [11] for solving the primal packing linear program with almost the same approximation factor as for the covering linear problem.

Therefore, our focus is on the following covering problem which is dual to the target-monitoring sensor network lifetime problem.

Target-Monitoring Sensor Covering Problem. Find minimum cost Steiner tree spanning the base and a set of sensors capable of simultaneous monitoring of all targets.

Consider a bipartite graph with vertex set $B = S \cup X$ and edges connecting sensors with visible targets. Then any set of sensors capable of simultaneous monitoring of all targets is a set of S -endpoints of a matching completely covering X . Therefore, all minimal feasible sets of sensors form a set of bases of a matroid or, in general, a polymatroid. The following problem generalizes the Steiner tree problem in a very natural way.

Polymatroid Steiner Problem (PSP). Given a graph $G = (V, E, c)$ with costs on edges and a polymatroid $P = P(V)$ on vertices of G , find minimum cost tree T within G spanning a base of P .

Equivalently, let $r : 2^V \rightarrow \{0, 1, \dots\}$ be a function on the set of vertices of G (called the rank function of the polymatroid $P(V)$) satisfying

- $r(A \cap B) + r(A \cup B) \leq r(A) + r(B)$, for all $A, B \subseteq V$ (submodularity)
- $r(\emptyset) = 0$
- if $A \subseteq B$ then $r(A) \leq r(B)$ (non-decreasing).

Then PSP asks for a minimum cost tree T spanning a maximum rank subset of V .

PSP generalizes various Steiner tree problem formulations. For example, setting the rank function $r(A) = |A \cap S|$, $A \subseteq V$ where $S \subseteq V$ is a given set of terminals, we obtain the classical *Steiner tree problem* which asks for a minimum cost tree spanning terminals S . The *group Steiner tree problem* searches a tree spanning at least one vertex from each of given groups (subsets of vertices) $V_1, \dots, V_k \subseteq V$ – it is also an instance of PSP with the rank function $r(A) = |\{V_i | A \cap V_i \neq \emptyset\}|$. The *covering Steiner tree problem* (see [16, 15, 8]) generalizes the group Steiner tree problem by requiring at least k_i vertices from a group V_i to be spanned – the corresponding rank function is $r(A) = \sum_{i=1}^k \min\{k_i, |A \cap V_i|\}$. Finally, the target-monitoring sensor covering problem is reduced to PSP by adding a single auxiliary target matching the base and setting $r(A)$ equal to the maximum number of targets that A can match.

The complexity of PSP can be derived from the recent papers [12, 13]. Halperin and Krauthgamer [13] showed that for every fixed $\epsilon > 0$, Group Steiner Tree problem admits no $\log^{2-\epsilon} n$ -approximation, unless NP has quasi-polynomial Las Vegas algorithms.

When applying primal-dual algorithm of [11] it is necessary to solve weighted target-monitoring sensor covering problem, i.e., the version in which each sensor has a certain weight and the cost of the solution is sum of weights of chosen sensors rather than just number of chosen sensors. PSP does not seem to generalize the node-weighted version, but it can be reduced to the following:

Polymatroid Directed Steiner Problem (PDSP). Given a directed graph $G = (V, E, c)$ with costs on edges and a polymatroid $P = P(V)$ on vertices of G . Find minimum cost arborescence T within G connecting a given root $r \in V$ to all vertices of at least one base of P .

The PDSP generalizes the Directed Steiner Tree Problem which can be obtained from PDSP by setting rank of a subset to its size. The best known approximation algorithm, due to Charikar et. al. [6], has running time $O(n^i k^{2i})$ and approximation ratio $i^2(i-1)k^{1/i}$ for any fixed $i > 1$. Thus, in polynomial time, their approximation ratio is $O(k^\epsilon)$, while in quasipolynomial time ($O(n^{c \lg n})$, for constant c) they achieve a polylogarithmic approximation ratio of $O(\log^3 k)$.

The simple energy model for the target-monitoring sensor network lifetime problem can be enhanced by allowing sensors to choose the power of transmission. Then the energy consumption of each sensor s_i is proportional to the cost of the hop connecting s_i to the next sensor on the path to the base as well as time t during which T is used. This model straightforwardly reduces the target-monitoring sensor network lifetime problem to PDSP. Note that the weighted version of PDSP is equivalent to PDSP – the cost of each edge e should be multiplied by the weight of the beginning of e .

In the next section we show how to generalize algorithm of [6] to solve PDSP with almost the same approximation factor. The section 3 is devoted to generalization of the algorithm of [7] to solve PSP with the polylogarithmic approximation ratio.

2 The Polymatroid Directed Steiner Tree Problem

In this section we establish performance bounds of the generalization of the algorithm of Charikar, Chekuri, Cheung, Dai, Goel, Guha, and Li [6] to the Polymatroid Directed Steiner Tree problem.

First we introduce a version of Polymatroid Directed Steiner Trees which allows the presentation of the algorithm. Without loss of generality, we assume that the directed graph is complete and $c(u, v)$ equals the minimum cost path from u to v . All the edges and trees in this section are directed. Given a set of nodes $X \subseteq V$, we denote by r_X the rank function with X contracted; that is $r_X(Z) = r(X \cup Z) - r(X)$. The rank function of a polymatroid is submodular implying that if $X \subseteq Y$, then for any T , $r_X(T) \geq r_Y(T)$. Let $PDST(k, v, X)$ denote the problem of finding the minimum-cost tree T rooted at v with $r_X(T) \geq k$. Note that the new version is in fact equivalent with the standard version as r_X is another submodular rank function. One can think of r_X as the residual rank function.

We ensure that all the nodes v with $r(v) > 0$ do not have outgoing edges by "duplicating" v as follows: if v has outgoing edges and positive rank, we introduce another node v' with $r(v') = 0$, replace every edge incident to v by a corresponding

Input: k, v, X

Output: An i -level tree $T = T_i(k, v, X)$ rooted at v with $r_X(T) \geq k$

-
0. Let $L(v)$ be the vertices reachable from v . If $r_X(L(v)) < k$, return \emptyset .
 1. If $i = 0$, return the tree with no edges and vertex set $\{v\}$ if $r_X(v) \geq k$, or \emptyset if $r_X(v) < k$.
 2. $j \leftarrow 1; k_j \leftarrow k; X_j \leftarrow X$
 3. while $k_j > 0$
 - 3.1 $T_{BEST}(j) \leftarrow \emptyset$
 - 3.2 for each vertex $u \in V$ and each $k', 1 \leq k' \leq k_j$
 - 3.2.1 $T' \leftarrow A_{i-1}(k', u, X_j) \cup \{(v, u)\}$
 - 3.2.2 if $d_{X_j}(T_{BEST}(j)) > d_{X_j}(T')$ then $T_{BEST}(j) = T'$
 - 3.3 $k_{j+1} \leftarrow k_j - r_{X_j}(T_{BEST}(j)); X_{j+1} \leftarrow X_j \cup L(T_{BEST}(j)); j \leftarrow j + 1$
 4. Return $\cup_{q=1}^{j-1} T_{BEST}(q)$
-

Fig. 1. Algorithm $A_i(k, v, X)$

edge incident to v' , and introduce the edge (v', v) of cost 0. This allows us to write $r(T) = r(V(T)) = r(L(T))$ for any directed tree T with vertex set $V(T)$ and leafs $L(T)$ assuming T 's root has rank 0 (as will be the case for all our trees: even the original root is duplicated if it has positive rank).

Let $c(T)$ be the cost of the directed tree T (the sum of the costs of the edges of T). Then we define the *density* of the tree T with respect to vertex set X as $d_X(T) = c(T)/r_X(T)$, where $V(T)$ is the vertex set of the tree and $L(T)$ is the vertex set of the leafs of the tree.

An l -level tree is a tree where no leaf is more than l edges away from the root. Robins and Zelikovsky [14] give:

Lemma 1. For all $l \geq 1$ and any tree $T \subseteq G$, there exists an l -level tree $T' \subseteq G$ with $L(T') = L(T)$ and $c(T') \leq l \cdot |L(T)|^{\frac{1}{l}} c(T)$.

An earlier claim from [18] that $c(T') \leq |L(T)|^{\frac{1}{l}}$, used in [6], has a gap in the proof.

2.1 The Algorithm

We describe the Charikar et. al. algorithm [6], generalized for our more general problem. The recursive algorithm $A_i(k, v, X)$ appears in Figure 1. The parameters passed down are the desired rank k , the desired root v , the maximum height i , and a pointer to a vector X describing the vertices already in the tree. The algorithm returns a pointer to an i -level tree $T = T_i(k, v, X)$ rooted at v satisfying $r_X(T) \geq k$, or \emptyset if no such tree exists. The base case is $i = 1$ (as opposed to $i = 2$ in the original version), as it is NP-Hard to compute a minimum density bunch (a tree with only one vertex with outdegree larger than one) in the polymatroid setting. When $i > 1$, the recursive algorithm copies the vector X and uses the copy during its execution, while when $i = 1$ the vector X is not modified.

Note the following invariant of the algorithm: $r_X(X_j) = k - k_j$. Indeed, $r_X(X_1) = r(X) - r(X) = 0 = k - k_1$, and $r_X(X_{j+1}) = r(X_{j+1}) - r(X) = r(X_{j+1}) - r(X_j) +$

$r(X_j) - r(X) = r_{X_j}(T_{BEST}(j)) + r_X(X_j) = (k_j - k_{j+1}) + (k - k_j) = k - k_{j+1}$. In particular, we have $r_X(T_i(k, v, X)) \geq k$, so the returned solution is valid.

Let $T_{OPT}^{(i)}(k, v, X)$ be an optimum i -level tree solving $PDST(k, v, X)$.

Lemma 2. *For all $i \geq 1$, each tree $T_{BEST}(j)$ chosen by algorithm $A_i(k, v, X)$ satisfies*

$$d_{X_j}(T_{BEST_j}) \leq i \cdot d_{X_j}(T_{OPT}^{(i)}(k_j, v, X_j))$$

Proof. The proof is by induction on i , with the base case $i = 0$ being immediate.

Assume the statement of the lemma holds for all k, v, X , and $i - 1$. $T_{OPT}^{(i)}(k_j, v, X_j)$ consists of several edges (v, u_p) and subtrees T_p rooted at u_p , such that $r_{X_j}(\cup_p T_p) \geq k_j$. Submodularity implies that $\sum_p r_{X_j}(T_p) \geq r_{X_j}(\cup_p T_p)$, and therefore by an averaging argument and renumbering, we have

$$\frac{c(v, u_1) + c(T_1)}{r_{X_j}(T_1)} \leq d_{X_j}(T_{OPT}^{(i)}(k_j, v, X_j)) \tag{1}$$

Consider the execution of the algorithm $A_{i-1}(k_j, u_1, X_j)$. Trees R_1, R_2, \dots are selected in this order, and let $Q_p = \cup_{q=1}^p R_q$. Now, when $A_{i-1}(k', u_1, X_j)$ is called for $k' = r_{X_j}(Q_p)$, then $Q_p \cup \{(v, u_1)\}$ is returned and is a candidate for $T_{BEST}(j)$. Our goal is to wisely choose such p and show that it has an appropriate density

$$\frac{c(v, u_1) + c(Q_p)}{r_{X_j}(Q_p)} \leq i \frac{c(v, u_1) + c(T_1)}{r_{X_j}(T_1)} \tag{2}$$

which together with Equation 1 would imply the theorem. We pick p to be the smallest integer such that $r_{X_j}(Q_p) \geq r_{X_j}(T_1)/i$, which implies

$$\frac{c(v, u_1)}{r_{X_j}(Q_p)} \leq i \frac{c(v, u_1)}{r_{X_j}(T_1)} \tag{3}$$

It remains to prove that

$$\frac{c(Q_p)}{r_{X_j}(Q_p)} \leq i \frac{c(T_1)}{r_{X_j}(T_1)} \tag{4}$$

as this equation together with Equation 3 implies Equation 2.

By the induction hypothesis, we have for all $q \leq p$

$$\frac{c(R_q)}{r_{Q_{q-1} \cup X_j}(R_q)} \leq (i - 1) \frac{c(T_1)}{r_{Q_{q-1} \cup X_j}(T_1)} \tag{5}$$

since R_q is $T_{BEST}(q)$ when executing $A_{i-1}(r_{X_j}(T_1), u_1, X_j)$. We picked p such that $r_{X_j}(Q_{p-1}) < \frac{1}{i} r_{X_j}(T_1)$, and therefore $r_{X_j \cup Q_{p-1}}(T_1) = r(X_j \cup Q_{p-1} \cup T_1) - r(X_j \cup Q_{p-1}) \geq r(X_j \cup T_1) - r(X_j \cup Q_{p-1}) = (r(X_j \cup T_1) - r(X_j)) - (r(X_j \cup Q_{p-1}) - r(X_j)) = r_{X_j}(T_1) - r_{X_j}(Q_{p-1}) \geq \frac{i-1}{i} r_{X_j}(T_1)$.

Submodularity of the rank function implies that for all $q \leq p$,

$$r_{X_j \cup Q_{q-1}}(T_1) \geq r_{X_j \cup Q_{p-1}}(T_1) \geq \frac{i-1}{i} r_{X_j}(T_1) \tag{6}$$

and, therefore,

$$\begin{aligned} \sum_{q=1}^p c(R_q) &\leq \sum_{q=1}^p r_{Q_{q-1} \cup X_j}(R_q)(i-1) \frac{c(T_1)}{r_{Q_{q-1} \cup X_j}(T_1)} \\ &\leq i \frac{c(T_1)}{r_{X_j}(T_1)} \sum_{q=1}^p r_{Q_{q-1} \cup X_j}(R_q) \end{aligned} \tag{7}$$

But

$$\begin{aligned} r_{X_j}(Q_p) &= r(X_j \cup Q_p) - r(X_j) \\ &= r(X_j \cup Q_p) - r(X_j \cup Q_{p-1}) + r(X_j \cup Q_{p-1}) - \dots - r(X_j \cup Q_0) \\ &= \sum_{q=1}^p r_{X_j \cup Q_{q-1}}(R_q) \end{aligned}$$

Thus Equation 7 implies Equation 4, finishing the proof of Lemma 2. ■

Theorem 1. For every $i > 1$, $k \geq 0$, $v \in V$ and $X \subseteq V$, the algorithm $A_i(k, v, X)$ provides an $i^3 k^{1/i}$ approximation to $PDST(k, v, X)$ in time $O(n^{i+1} k^{2i+2} q(n))$, where $q(n)$ is the time an oracle returns $r_X(v)$ for an arbitrary $X \subseteq V$.

Proof. The proof follows [6]. Note that $T_{OPT}(k, v, X)$ is a valid solution for $PDST(k_j, v, X_j)$ since

$$\begin{aligned} r_{X_j}(T_{OPT}(k, v, X)) &= r(X_j \cup T_{OPT}(k, v, X)) - r(X_j) \\ &\geq r(X \cup T_{OPT}(k, v, X)) - r(X) - (r(X_j) - r(X)) \\ &\geq k - r_X(X_j) = k - (k - k_j) = k_j \end{aligned}$$

where we used the invariant of the procedure $A_i(k, v, X)$. Therefore, by Lemma 1, we have $c(T_{OPT}^{(i)}(k_j, v, X_j)) \leq i k_j^{1/i} c(T_{OPT}(k_j, v, X_j))$ and $c(T_{OPT}(k_j, v, X_j)) \leq c(T_{OPT}(k, v, X))$, and by Lemma 2, we have

$$\begin{aligned} d_{X_j}(T_{BEST}(j)) &\leq i \cdot d_{X_j}(T_{OPT}^{(i)}(k_j, v, X_j)) \\ &\leq i \cdot i \cdot k_j^{1/i} \cdot \frac{c(T_{OPT}(k, v, X))}{k_j} \end{aligned}$$

So we know that

$$\frac{c(T_{BEST}(j))}{r_{X_j}(T_{BEST}(j))} \leq i^2 k_j^{1/i} \frac{c(T_{OPT}(k, v, X))}{k_j}$$

Since $r_{X_j}(T_{BEST}(j)) = k_{j+1} - k_j$, we obtain

$$c(T_{BEST}(j)) \leq i^2 c(T_{OPT}(k, v, X)) (k_{j+1} - k_j) \frac{k_j^{1/i}}{k_j}$$

Summing over j (cf. Lemma 1 of [6]) and using $k_1 = k$ and $k_{j+1} < 0$, we obtain

$$\begin{aligned}
 c(T_i(k, v, X)) &\leq i^2 c(T_{OPT}(k, v, X)) \sum_j (k_{j+1} - k_j) \frac{k_j^{1/i}}{k_j} \\
 &\leq i^2 c(T_{OPT}(k, v, X)) \int_0^k x^{1-1/i} dx \\
 &= i^3 k^{1/i} c(T_{OPT}(k, v, X))
 \end{aligned}$$

The procedure A_i invokes A_{i-1} at most nk^2 times, and the bound on the running time follows. ■

If the input is a star (directed tree with one level), the algorithm becomes the Greedy algorithm for the Submodular Set Covering problem [17].

Corollary 1. *The approximation ratio of the Greedy Algorithm applied to the Submodular Set Covering problem is at most $1 + \ln k$.*

Wolsey [17] investigated this problem and has shown that the Greedy algorithm has in fact a slightly better approximation ratio H_q , where $q = \max_{v \mid r(v) > 0} r(v)$.

3 The Polymatroid Steiner Tree Problem

In this section we give the solution of the (undirected) Polymatroid Steiner tree Problem. The first choice for solving this problem is to generalize the linear-program based algorithms of Garg, Konjewood, and Ravi [10], Konjewood, Ravi, and Srinivasan [15], and Zosin and Khuller [19]. Unfortunately, it is truly cumbersome to apply linear programs to PSP. Instead, our algorithm for PSP mostly relies on the combinatorial approximation algorithm for Group Steiner Tree of Chekuri, Even, and Korsatz [7].

The Chekuri et. al. algorithm is obtained by modifying the Charikar et. al. algorithm [6], and is applied after the graph metric has been replaced by a tree metric [1, 2, 5, 9], losing a factor of $O(\log n)$ in approximation ratio. Thus we also assume the input is a rooted tree.

Chekuri et. al. [7] preprocess the tree to decrease the depth and degree. This preprocessing approximately preserves the cost of any solution, so we can apply it Polymatroid Steiner Tree as well. Precisely, every set of leaves $L \subseteq L(T)$ induces a subtree T_L consisting of the union of all paths from the root to the leaves in L . If A and B are two trees with the same root and the same set of leaves, the tree B is a α -faithful representation of the tree A if

$$\forall L \subseteq L(A) : c(A_L) \leq c(B_L) \leq \alpha c(A_L).$$

The preprocessing is stated in the following theorem of Chekuri et. al. [7]:

Theorem 2. *Given a tree T with n leaves and integer parameters α and $\beta > 2$, there is a linear time algorithm to transform T into a $O(\alpha)$ -faithful tree T' with height $O(\log_\alpha n + \log_{\beta/2} n)$ and maximum degree $O(\beta)$.*

Thus, by losing a factor of $O(\alpha \log n)$ ($\log n$ comes from embedding the original metric in a tree metric), we can assume that the instance for Polymatroid Steiner Tree is a tree with height $O(\log_\alpha n + \log_{\beta/2} n)$ and maximum degree $O(\beta)$.

For the reader familiar with the Chekuri et. al. paper [7], below you'll find the correspondence between the notions used by this previous paper and our notions. Their theorems and proofs "translate" to PST, some of them directly.

- $w(T) \text{ --- } c(T)$
- $z' \text{ --- } k$
- $r' \text{ --- } v$
- $T_{r'}$ (excluding the leaves already reached) --- X (the leaves already reached)
- $T_{aug} \text{ --- } T_{BEST}(j)$
- $z^{res} \text{ --- } k_j$
- $\gamma(T) \text{ --- } d_X(T)$
- $cover \text{ --- } \cup_{i=1}^j T_{BEST}(j)$
- $m(T) \text{ --- } r_{X_j}(T)$
- $m(cover) \text{ --- } r_{X_j} X_j$
- remove group covered by T_{aug} from $T^{res} \text{ --- } X_{j+1} \leftarrow X_j \cup L(T_{BEST}(j))$
- $cover_h \text{ --- } j_h$
- **Group-Steiner*** (T^{res}, z^{res}) --- $T_{OPT}(k_j, v, X_j)$

The Modified-Group-Steiner algorithm of Chekuri et. al. [7], as generalized for Polymatroid Steiner Tree for trees is described in Figure 2. The recursive procedure $M(k, v, X)$ uses an integer parameter λ as the basis for the geometric search. We use T_v to denote the subtree rooted at v and C_v the set of children of v . $h(T')$ denotes the height of a subtree T' .

In the practical implementations, one can continue the while loop from Step 3 instead of stopping in Step 3.4, returning the lowest density tree from $\cup_{q=1}^{j-1} T_{BEST}(q)$ and $\cup_{q=1}^{j_h-1} T_{BEST}(q)$, where j_h is the value of j at the first moment $r_X(X_j) \geq k/(h(T_v)+1)$.

The proof of the following lemma can be directly generalized from the proof of Lemma 3.3 of [7]. We omit details but notice that in the base case (leaves) a polymatroid oracle call must be made, which is assumed to take time $q(n)$.

Input: k, v, X

Output: A tree $T = T(k, v, X)$ rooted at v with $r_X(T) \geq k$, or \emptyset if no such tree exists

0. If $r_X(L(T_v)) < k$, return \emptyset .
1. If v is a leaf, return the tree with no edges and vertex set $\{v\}$
2. $j \leftarrow 1; k_j \leftarrow k; X_j \leftarrow X$
3. while $k_j > 0$
 - 3.1 $T_{BEST}(j) \leftarrow \emptyset$
 - 3.2 for each vertex $u \in C_v$ and each k' power of $(1 + \lambda)$ in $\left[\frac{k_j}{deg(v)(1+1/\lambda)(1+\lambda)}, k_j \right]$
 - 3.2.1 $T' \leftarrow M(k', u, X_j) \cup \{(v, u)\}$
 - 3.2.2 if $d_{X_j}(T_{BEST}(j)) > d_{X_j}(T')$ then $T_{BEST}(j) = T'$
 - 3.3 $k_{j+1} \leftarrow k_j - r_{X_j}(T_{BEST}(j)); X_{j+1} \leftarrow X_j \cup L(T_{BEST}(j)); j \leftarrow j + 1$
 - 3.4 If $r_X(X_j) \geq k/(h(T_v) + 1)$, then return $\cup_{q=1}^{j-1} T_{BEST}(q)$

Fig. 2. Algorithm $M(k, v, X)$

Lemma 3. *Let Δ be the maximum degree of the tree and $b = \Delta(1 + 1/\lambda)(1 + \lambda)$. The running time of $M(k, v, X)$ is $O((n + q(n))\alpha^{h(T_v)})$ where $\alpha = b \cdot h(T_v) \cdot \log k \cdot \Delta \cdot \log_{1+\lambda} b$.*

The main lemma needed for establishing the approximation ratio is a generalization of Lemma 3.4 of [7].

Lemma 4.

$$d_{X_j}(T_{BEST}(j)) \leq (1 + \lambda)^{2h(T_v)} h(T_v) d_{X_j}(T_{OPT}(k_j, v, X_j))$$

Proof. In general, the proof follows [7]. We use $\gamma^* = d_{X_j}(T_{OPT}(k_j, v, X_j))$. The proof is by induction on $h(T_v)$, the height of the subtree rooted at v . Both the base case $h(T_v) = 1$ and the general case need the following argument.

Let u_1, u_2, \dots, u_d be the children of v in $T_{OPT}(k_j, v, X_j)$, and $T_i, 1 \leq i \leq d$, be the subtree of $T_{OPT}(k_j, v, X_j)$ rooted at u_i . Thus $r_{X_j}(\cup_{i=1}^d T_i) \geq k_j$. We divide the set $1, 2, \dots, d$ into the set B giving ‘‘big’’ trees: those i with $r_{X_j}(T_i) \geq \frac{k_j}{deg(v)(1+1/\lambda)}$, and the set S giving ‘‘small’’ trees: those i with $r_{X_j}(T_i) < \frac{k_j}{deg(v)(1+1/\lambda)}$. Then $r_{X_j}(\cup_{i \in S} T_i) \leq \sum_{i \in S} r_{X_j}(T_i) < \frac{k_j}{1+1/\lambda}$ and, therefore,

$$\begin{aligned} \sum_{i \in B} r_{X_j}(T_i) &\geq r_{X_j}(\cup_{i \in B} T_i) \\ &\geq k_j - d \frac{k_j}{deg(v)(1 + 1/\lambda)} \\ &\geq \frac{k_j}{1 + \lambda} \end{aligned}$$

By an averaging argument, there is a big tree (which we renumber T_1) such that

$$\begin{aligned} \frac{c(v, u_1) + c(T_1)}{r_{X_j}(T_1)} &\leq \frac{\sum_{i \in B} c(v, u_i) + c(T_i)}{\sum_{i \in B} r_{X_j}(T_i)} \\ &\leq (1 + \lambda) \frac{c(T_{OPT}(k_j, v, X_j))}{k_j} \\ &= (1 + \lambda)\gamma^* \end{aligned} \tag{8}$$

Let z be the power of $(1 + \lambda)$ such that $z \leq r_{X_j}(T_1) < (1 + \lambda)z$ and note that z is in the range of powers of $(1 + \lambda)$ considered in Line 3.2 of the algorithm. Therefore $M(z, u_1, X_j)$ is called.

In the base case, u_1 is a leaf, and therefore a candidate for $T_{BEST}(j)$ is the tree T' with only the edge (v, u_1) , having density $d_{X_j}(T') = \frac{c(v, u_1)}{r_{X_j}(T_1)} \leq (1 + \lambda)\gamma^*$. Thus the base case of induction holds.

In the general case, let R_1, R_2, \dots, R_p be the trees picked as T_{BEST} during the execution of $M(z, u_1, X_j)$. Let $Q_q = \cup_{i=0}^q R_i$ and let $h_1 = h(T_{u_1})$. Induction gives for $i \in \{1, 2, \dots, p\}$

$$\begin{aligned} \frac{c(R_i)}{r_{X_j \cup Q_{i-1}}(R_i)} &\leq (1 + \lambda)^{2h_1} h_1 d_{X_j \cup Q_{i-1}}(T_{OPT}(z - r_{X_j}(Q_{i-1}), u_1, X_j \cup Q_{i-1})) \\ &\leq (1 + \lambda)^{2h_1} h_1 \frac{c(T_1)}{r_{X_j \cup Q_{i-1}}(T_1)} \end{aligned} \tag{9}$$

where the second inequality follows from the fact that $r_{X_j \cup Q_{i-1}}(T_1) \geq z - r_{X_j}(Q_{i-1})$ (which follows from $r_{X_j}(T_1) \geq z$ and the submodularity of r), and, therefore, T_1 is a candidate for $T_{OPT}(z, u_1, X_j \cup Q_{i-1})$.

By the return condition of the algorithm, $r_{X_j}(Q_{i-1}) < \frac{1}{h_1+1}z$ and, therefore,

$$\begin{aligned} r_{X_j \cup Q_{i-1}}(T_1) &\geq r_{X_j}(T_1) - r_{X_j}(Q_{i-1}) \\ &\geq z(1 - \frac{1}{h_1 + 1}) \\ &= \frac{h}{h_1 + 1}z \end{aligned}$$

Together with Equation 9, we obtain:

$$\begin{aligned} \frac{c(R_i)}{r_{X_j \cup Q_{i-1}}(R_i)} &\leq (1 + \lambda)^{2h_1} (h_1 + 1) \frac{c(T_1)}{z} \\ &\leq (1 + \lambda)^{2h_1+1} (h_1 + 1) \frac{c(T_1)}{r_{X_j}(T_1)} \end{aligned} \tag{10}$$

since z was chosen such that $z \leq r_{X_j}(T_1) < (1 + \lambda)z$. Thus,

$$\sum_{q=1}^p c(R_q) \leq \sum_{q=1}^p r_{X_j \cup Q_{q-1}}(R_q) (1 + \lambda)^{2h_1+1} (h_1 + 1) \frac{c(T_1)}{r_{X_j}(T_1)} \tag{11}$$

But

$$\begin{aligned} r_{X_j}(Q_p) &= r(X_j \cup Q_p) - r(X_j) \\ &= r(X_j \cup Q_p) - r(X_j \cup Q_{p-1}) + r(X_j \cup Q_{p-1}) - \dots - r(X_j \cup Q_0) \\ &= \sum_{q=1}^p r_{X_j \cup Q_{q-1}}(R_q) \end{aligned}$$

and, therefore,

$$\frac{\sum_{q=1}^p c(R_q)}{r_{X_j}(Q_p)} \leq (1 + \lambda)^{2h_1+1} (h_1 + 1) \frac{c(T_1)}{r_{X_j}(T_1)} \tag{12}$$

Note that p was picked such that $r_{X_j}(Q_p) \geq \frac{1}{h_1+1}z \geq \frac{1}{(h_1+1)(1+\lambda)}r_{X_j}(T_1)$ and therefore

$$\frac{c(v, u_1)}{r_{X_j}(Q_p)} \leq (h_1 + 1)(1 + \lambda) \frac{c(v, u_1)}{r_{X_j}(T_1)} \tag{13}$$

Combining the previous equation with Equation 12 we obtain

$$\frac{c(u, v) + \sum_{q=1}^p c(R_q)}{r_{X_j}(Q_p)} \leq (1 + \lambda)^{2h_1+1} (h_1 + 1) \frac{c(v, u_1) + c(T_1)}{r_{X_j}(T_1)} \quad (14)$$

Using Equation 8 and the fact that $h(T_v) \geq h(T_{u_1}) + 1 = h_1 + 1$, we obtain

$$\frac{c(u, v) + \sum_{q=1}^p c(R_q)}{r_{X_j}(Q_p)} \leq (1 + \lambda)^{2h(T_v)} h(T_v) \gamma^* \quad (15)$$

Thus, the tree T' returned by $M(z, u_1, X_j)$, which is a candidate for $T_{BEST}(j)$ in the execution of $M(k, v, X)$, satisfies $d_{x_j}(T') \leq (1 + \lambda)^{2h(T_v)} h(T_v) \gamma^*$. ■

Chekuri et. al. [7] choose (and we do the same) $\alpha = (\log n)^\epsilon$, $\beta = \log n$, $1/\lambda = \log n$. Assuming the oracle computation is polynomial time, the proof of Theorem 3.5 and Corollary 3.6 of [7] gives our corresponding statement:

Theorem 3. *There is a combinatorial polynomial-time $O(\frac{1}{\epsilon} \cdot \frac{1}{\log \log n} \cdot (\log n)^{1+\epsilon} \log k)$ -approximation algorithm for Polymatroid Steiner Tree on trees with n nodes, where k is the desired rank. For general undirected graphs, there is a combinatorial polynomial-time $O(\frac{1}{\epsilon} \cdot \frac{1}{\log \log n} \cdot (\log n)^{2+\epsilon} \log k)$ -approximation algorithm*

4 Conclusion

Motivated by applications in wireless sensor networks (when sensors can monitor only a single target), we have introduced the Polymatroid (Directed and Undirected) Steiner Tree Problems (PSP). These problems asks for a (directed) Steiner tree spanning a subset of terminals of sufficiently large rank. The undirected version of PSP generalizes all known versions of the Group Steiner Tree Problem and is shown to be solved by a generalization of the algorithm from [7] with the polylogarithmic approximation ratio. The directed version of PSP is a generalization of the Directed Steiner Tree Problem as well as Polymatroid Set Cover Problem. We show that this problem can be approximately solved by methods generalizing [6].

References

1. Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *FOCS*, 1996.
2. Y. Bartal. On approximating arbitrary metrics by tree metrics. In *STOC*, 1998.
3. P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. In *Proc. IEEE Wireless Communications and Networking Conference*, 2004.
4. G. Calinescu, S. Kapoor, A. Olshevsky, and A. Zelikovsky. Network lifetime and power assignment in ad-hoc wireless networks. In *Proc. 11th European Symposium on Algorithms*, 2003.

5. M. Charikar, C. Chekuri, A. Goel, and S. Guha. Approximating a finite metric by a small number of tree metrics. In *STOC*, 1999.
6. Moses Charikar, Chandra Chekuri, Toyat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation Algorithms for Directed Steiner Problems. *Journal of Algorithms*, 33(1):73–91, 1999.
7. C. Chekuri, G. Even, and G. Kortsarz. A combinatorial approximation algorithm for the group Steiner problem, 2002. Submitted for publication. Available on the web.
8. G. Even, G. Kortsarz, and W. Slany. On network design: fixed charge flows and the covering Steiner problem. In *SWAT 2002*, pages 318–329.
9. Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*, 2003.
10. N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.
11. Naveen Garg and Jochen Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.
12. E. Halperin, G. Kortsarz, R. Krauthgamer, A. Srinivasan, and N. Wang. Integrality ratio for Group Steiner Trees and Directed Steiner Trees. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 275–284, January 2003.
13. E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 585–594, 2003.
14. C. H. Helvig, G. Robins, and A. Zelikovsky. Improved approximation scheme for the group Steiner problem. *Networks*, 37(1):8–20, 2001.
15. G. Konjevod, R. Ravi, and A. Srinivasan. Approximation algorithms for the covering Steiner problem. *Random Structures and Algorithms*, 20(3):465–482, 2002. Preliminary version by Konjevod and Ravi in *SODA 2000*.
16. Goran Konjevod and R. Ravi. An approximation algorithm for the covering steiner problem. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 338–344. Society for Industrial and Applied Mathematics, 2000.
17. L.A. Wolsey. Analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2:385–392, 1982.
18. A. Zelikovsky. A series of approximation algorithms for the acyclic directed Steiner tree problem. *Algorithmica*, 18, 1997.
19. Leonid Zosin and Samir Khuller. On directed Steiner trees. In *SODA*, pages 59–63, 2002.

Geometric Optimization Problems Over Sliding Windows^{*}

Timothy M. Chan and Bashir S. Sadjad

School of Computer Science,
University of Waterloo,
Waterloo, Ontario, N2L 3G1, Canada
{tmchan, bssadjad}@uwaterloo.ca

Abstract. We study the problem of maintaining a $(1+\epsilon)$ -factor approximation of the *diameter* of a stream of points under the *sliding window* model. In one dimension, we give a simple algorithm that only needs to store $O(\frac{1}{\epsilon} \log R)$ points at any time, where the parameter R denotes the “spread” of the point set. This bound is optimal and improves Feigenbaum, Kannan, and Zhang’s recent solution by two logarithmic factors. We then extend our one-dimensional algorithm to higher constant dimensions and, at the same time, correct an error in the previous solution. In high nonconstant dimensions, we also observe a constant-factor approximation algorithm that requires sublinear space. Related optimization problems, such as the *width*, are also considered in the two-dimensional case.

1 Introduction

In conventional settings an algorithm has access to the whole input at once. In the *sliding window* model [12, 15], however, data elements come in one by one. In this model, we want to maintain some function over the input (for example, statistical information) but because of space limitations, we are not allowed to store all data in memory. In the more general *sliding window* model, the setting is similar but the goal function should be maintained over a window containing the N newest elements in the stream. In some cases the value of N is fixed in advance, but in some applications the size of the window may be dynamic. (For example, we may want to maintain the goal function over data received within the last one hour.)

The objective of this paper is to study some well-known optimization problems in computational geometry, *diameter*, *width*, and the *width*, under the sliding window model. For a set P of n points in d -dimensional space, the diameter $\Delta(P)$ is the distance between the furthest pair of points in P . The width of P is the minimum distance between two parallel hyperplanes where all points lie between them. We focus primarily on the case where d is a small constant.

^{*} Work done by the first author was supported by an NSERC Research Grant and a Premiere’s Research Excellence Award. This work has appeared as part of the second author’s Master’s thesis.

The diameter problem has been studied extensively in the traditional model. Although $O(n \log n)$ algorithms have been given for $d = 2$ [16] and $d = 3$ [8, 17], only slightly subquadratic algorithms are known for higher dimensions. This has prompted researchers [2, 4, 6] to consider approximation algorithms. For example, with the latest algorithm [7], a $(1 + \epsilon)$ -factor approximation of the diameter can be found in $O(n + (\frac{1}{\epsilon})^{d-1.5})$ time. Earlier Agarwal et al. [2] observed a simple approximation algorithm that runs in $O((\frac{1}{\epsilon})^{(d-1)/2}n)$ time and is in fact a data-stream algorithm requiring just $O((\frac{1}{\epsilon})^{(d-1)/2})$ space. Other data-stream approximation algorithms in the two-dimensional case were considered by Feigenbaum et al. [10] and Hershberger and Suri [13]. (No efficient data-stream exact algorithm is possible even in one dimension [10].) For the high-dimensional case, where d is not a fixed constant, Goel et al. [11] have proposed a $(1 + \epsilon)$ -approximation algorithm in the traditional model that runs in $O(n^{1+1/(1+\epsilon)} + dn)$ time, and Indyk [14] has developed a $(c + \epsilon)$ -approximation algorithm in the data stream model with sublinear space complexity $O(dn^{1/(c^2-1)} \log n)$ for $c > \sqrt{2}$.

The width problem has also been extensively studied in the traditional model, both in terms of exact algorithms [3, 16] and approximation algorithms [6, 9]. The recent approximation algorithm by Chan [7], for example, runs in $O(n + (\frac{1}{\epsilon})^{d-1})$ time for any fixed d . An efficient approximation algorithm in the data stream model has also been proposed by Chan, using only $O((\frac{1}{\epsilon} \log \frac{1}{\epsilon})^{d-1})$ space.

The previous (and perhaps the only) approximate diameter algorithm in the sliding window model was proposed by Feigenbaum et al. [10]. Their one-dimensional algorithm (briefly reviewed in Section 2) requires $O(\frac{1}{\epsilon} \log^3 N (\log R + \log \log N + \log \frac{1}{\epsilon}))$ bits of space, where N is the size of the sliding window and R is the spread of the points in the window; the spread is defined as the ratio between the diameter and the minimum non-zero distance between any two points in the window. Also Feigenbaum et al. have shown a lower bound of $\Omega(\frac{1}{\epsilon} \log R \log N)$ bits of space. They also claimed that their one-dimensional algorithm can be used to approximate the diameter in any fixed dimension d , with $O((\frac{1}{\epsilon})^{(d+1)/2} \log^3 N (\log R + \log \log N + \log \frac{1}{\epsilon}))$ bits of space.

To our knowledge, there is no existing solution for the width problem in the sliding window model.

In this paper, we give a number of new results in the sliding window model:

- For diameter in one dimension (Section 3), we present a simpler and more efficient algorithm that stores only $\Theta(\frac{1}{\epsilon} \log R)$ points. Under the assumption (made also by Feigenbaum et al.) that coordinates can be encoded with $O(\log N)$ bits, our algorithm thus uses $O(\frac{1}{\epsilon} \log R \log N)$ bits of space, matching the known lower bound. Our algorithm is also faster, requiring $O(1)$ time per point (Feigenbaum et al.’s method requires $O(\log N)$ time per point).
- For diameter in a fixed dimension (Section 4), we point out a small error in Feigenbaum et al.’s previous solution. In addition, we show that our one-dimensional algorithm can be extended to higher dimensions. The number of points stored is $O((\frac{1}{\epsilon})^{(d+1)/2} \log \frac{R}{\epsilon})$, which again improves the previous (claimed) bound.

- For diameter in a higher non-constant dimension, or more generally, in a metric space (Section 5), we mention a $(6 + \epsilon)$ -approximation algorithm that uses sublinear space. (Although a constant-factor result is trivial for the data stream model, it is not as obvious for sliding windows.)
- We also give the first solution for the width problem in the sliding window model in the two-dimensional case (Section 6). More generally, we show how to maintain an ϵ -approximation [1]. Although the space requirement of this method is dependent on a new variable R' , defined as the ratio between diameter and the minimum width of a consecutive triple of points, we show such a dependence is necessary in the sliding window model.

2 Diameter in One Dimension: The Previous Algorithm

In this section, we briefly review the previous technique of Feigenbaum et al. [10] for maintaining the diameter of a one-dimensional point set over a sliding window of size at most N , as the ideas here will be helpful later (in Sections 5 and 6). At any particular time we have a set of points (or numbers) P in the sliding window, where an arrival time is associated to each point. We say that p is newer than q if the arrival time of p is greater than the arrival time of q (older otherwise). We want to approximate the diameter $\Delta(P) = \max_{p,q \in P} \|p - q\|$, where $\|p - q\|$ denotes the distance between two points p and q .

The basic approach is similar to the *cluster-based approach* of Bentley and Saxe [5] (which was also used in some previous data stream algorithms, e.g., [1]). The input is split into several clusters. Each cluster represents an interval of time and the size of each cluster is a power of two. In each cluster C , a small subset $N_C \subseteq C$ of points (called *representative points*) is recorded as an approximation of C and obeys the following rules:

1. For each cluster C , the exact location of the newest point o_C is recorded as the *center* of C .
2. Each new point p forms a cluster C of size one.
3. For each point p in cluster C there exists $q \in N_C$ such that both p and q are on one side of o_C , q is not older than p , and

$$\frac{1}{1 + \epsilon} \|o_C - q\| \leq \|o_C - p\| \leq (1 + \epsilon) \|o_C - q\|; \tag{1}$$

we say that q is a *representative point* of p (or p is *close* to q).

4. When there are more than two clusters of size k , the two older clusters C_1 and C_2 are combined to form a cluster C of size $2k$ and a merge procedure is applied to form N_C from N_{C_1} and N_{C_2} .

To construct a subset N_C satisfying Inequality 1 from C , the basic strategy is as follows: let o be the newest point in C and let δ be the distance of the closest point in C from o . For each $i \geq 0$, find the newest point at distance between $(1 + \alpha)^i \delta$ and $(1 + \alpha)^{i+1} \delta$ (for an α to be determined later) and put this point in N_C .

To merge two clusters C_1 and C_2 , the idea is to apply the above procedure to compute N_C not from C but from just the points in $N_{C_1} \cup N_{C_2}$ (ignoring details about what δ should be for N_C).

A problem that arises is that each merge may cause an additional additive error of $\alpha\Delta(P)$. In other words, p might be rounded to q and then after a merge, q might be rounded to another point r . To overcome this problem, the idea is to use $\alpha = \frac{\epsilon}{\log N}$. Since at most $\log N$ merges can happen to a point p , it always has a representative q with $\|q - p\| \leq (\log N)\alpha\Delta(P) = \epsilon\Delta(P)$.

In each cluster C , we keep $O(\log_{1+\alpha} R) = O(\frac{1}{\epsilon} \log N \log R)$ points in N_C , and there are $O(\log N)$ clusters, so the space complexity of this method is about the space needed for $O(\frac{1}{\epsilon} \log^2 N \log R)$ points, i.e., about $O(\frac{1}{\epsilon} \log^3 N \log R)$ bits.

3 Diameter in One Dimension: The New Algorithm

We now present a more direct algorithm for one-dimensional diameter that avoids working with multiple clusters. To find the furthest pair in P , it is enough to find the largest and smallest numbers, and we do this by two similar data structures that maintain an approximate maximum and minimum of the points. An “approximate maximum” here refers to a $q \in P$ such that $\|q - p\| \leq \epsilon\Delta(P)$ where $p \in P$ is the maximum point.

We now describe a data structure that maintains a subset of $O(\log_{1+\epsilon} R)$ points of P , the largest of which is an approximate maximum. The data structure supports insertion of a new point and deletion of the oldest point. The key idea is in the following definition:

Definition 1. $Q = \langle q_1, q_2, \dots, q_k \rangle$ is a summary sequence of P if

- $q_1 < q_2 < \dots < q_k$ and $q_k \in P$.
- Q is a summary sequence of P if $q_i \in P$ for all i .
- Q is a summary sequence of P if $\|p - q_i\| \leq \epsilon\Delta_p(P)$ for all i and $p \in P$.

Let us see why summary sequences are important. First notice that upon insertion of a new point p , we can delete all points $b \in P$ that are not greater than p . This is because from now on p is a better approximation than b for the maximum value. So, condition 1 can be guaranteed. Conditions 2 and 3 ensure that $\text{pred}_Q(p)$ or $\text{succ}_Q(p)$ can be used as a “representative” of p .

Notice that the summary sequence is not unique. For example in Figure 1, from the summary sequence a_1, a_2, \dots, a_7 , we can get another summary sequence by deleting a_3, a_4 , and a_5 while ensuring condition 3. The interesting question is to how to maintain small summary sequences.

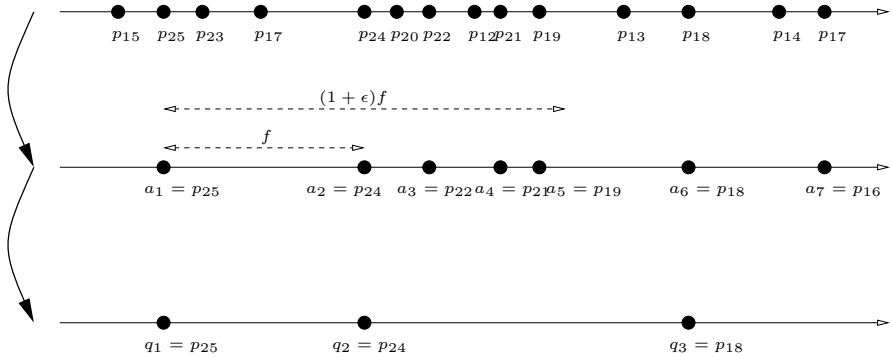


Fig. 1. An example of a summary sequence. (The index of each point is equal to its arrival time.)

Our algorithm to maintain a summary sequence Q is remarkably simple and is described completely in the pseudocode below. We assume that Q is stored in a doubly linked list.

Insert(p): /* given a new point p^* /

1. Remove all points in Q that are less than p , and put p at the beginning.
2. After every $\frac{1}{\epsilon} \log R$ insertions, run Refine.

Refine:

1. Let q_1 and q_2 be the first and the second points in Q respectively.
2. Let $q := q_2$.
3. **While** q is not the last element of Q **do**
4. Let x and y be the elements before and after q in Q .
5. **If** $(1 + \epsilon) \|q_1 - x\| \geq \|q_1 - y\|$ **then** remove q from Q .
6. Continue with q equal to y .

Delete(p): /* given the oldest point p^* /

1. Just remove p from Q if $p \in Q$.

We prove the correctness of our algorithm:

Lemma 1. *Let P be a set of n points in the plane, and let Q be the summary sequence of P maintained by our algorithm. Then $|Q| = O(\frac{1}{\epsilon} \log R)$.*

Proof: We show that conditions 1–3 remain valid each time a single point is removed from Q . This is obvious for condition 1. Regarding condition 2, just note that the new predecessor of a point is not older than the previous predecessor by condition 1.

We now consider condition 3. Let $p \in P$. Before the removal, we have either $\|p - \text{pred}_Q(p)\| \leq \epsilon \Delta_p(P)$ or $\text{succ}_Q(p)$ is newer than p . Set $q = \text{pred}_Q(p)$ in the former case, and $q = \text{succ}_Q(p)$ in the latter. If q is not the point being removed, then the condition clearly remains true for the point p . Otherwise, by

the design of the refine algorithm, the predecessor x and successor y of q must satisfy $(1 + \epsilon)\|q_1 - x\| \geq \|q_1 - y\|$. This implies that $\|p - x\| \leq \|y - x\| \leq \epsilon\|x - q_1\| \leq \epsilon\Delta_p(P)$, because x and q_1 are newer than q by condition 1. Since x is the new predecessor of p , condition 3 remains true for the point p .

For the second part of the lemma, let $Q = \langle q_1, q_2, \dots, q_k \rangle$ after the refine algorithm. Then for each $1 < i < k - 2$ we have $\|q_1 - q_{i+2}\| > (1 + \epsilon)\|q_1 - q_i\|$, because otherwise we would have removed q_{i+1} . Since $\|q_1 - q_2\|$ is at least the minimum non-zero distance $\delta(P)$, and $\|q_1 - q_k\|$ is at most the maximum distance $\Delta(P)$, we have $k \leq 2 \log_{1+\epsilon} \frac{\Delta(P)}{\delta(P)} \leq 2 \log_{1+\epsilon} R = O(\frac{1}{\epsilon} \log R)$. \square

It is easy to check that conditions 1-3 remain true after line 1 of the insert algorithm, or after a deletion.

Theorem 1. $\dots O(\frac{1}{\epsilon} \log R) \dots$
 $(1 + \epsilon) \dots O(1) \dots$
 $\dots O(1) \dots$

Proof: The number of points in the summary sequence is $O(\frac{1}{\epsilon} \log R)$ after each refine, and we do refine after each $\frac{1}{\epsilon} \log R$ insertions. Thus, the space complexity of this method is $O(\frac{1}{\epsilon} \log R)$. Upon insertion of a new point, we may remove $O(\frac{1}{\epsilon} \log R)$ points from the current summary sequence, but each point will be removed at most once. On the other hand, an execution of the refine algorithm requires $O(\frac{1}{\epsilon} \log R)$ time but is done once per $\frac{1}{\epsilon} \log R$ insertions. Thus, the amortized insertion time is $O(1)$. \square

The algorithm can be modified by standard techniques so that the worst-case insertion time is $O(1)$.

Besides being space-optimal (assuming coordinates have $O(\log N)$ bits), a desirable new feature of our data structure is that the size of the window need not be known in advance and can change dynamically (i.e., each insertion does not have to be followed by one deletion, or vice versa).

4 Diameter in Higher Fixed Dimensions

Feigenbaum et al. have extended their one-dimensional diameter algorithm to any fixed dimensions. They use the following well-known lemma:

Lemma 2. $L = \Theta((\frac{1}{\epsilon})^{(d-1)/2}) \dots \mathbb{R}^d \dots$
 $x \in \mathbb{R}^d \dots x \dots \ell \in L \dots \arccos(\frac{1}{1+\epsilon}) \dots$

Such a set L can be constructed in $\Theta(|L|)$ time, for example, by a grid method [7]. The idea is to construct this set L and use the one-dimensional structure to maintain the diameter of the projection of the point set to each line $\ell \in L$.

Lemma 3. Let L be a line in \mathbb{R}^d . Let P be a point set in \mathbb{R}^d . Let \mathcal{B} be a set of $(1 + \epsilon)$ balls of radius Δ centered at points in P . Let $\ell \in L$ be a line. Let \mathcal{B}' be a set of $(1 + O(\epsilon))$ balls of radius Δ centered at points in P .

Proof: Let $p, q \in P$ be the furthest pair of points in P . Suppose the angle between line \overrightarrow{pq} and $\ell \in L$ is at most $\arccos(\frac{1}{1+\epsilon})$. Let p' and q' be the projection of p and q on ℓ . Then $\|p - q\| \leq (1 + \epsilon)\|p' - q'\|$. On the other hand, the maximum returned value is at least $\frac{1}{1+\epsilon}\|p' - q'\| \geq \frac{1}{(1+\epsilon)^2} \Delta$. \square

Feigenbaum et al. did not observe the following problem: If we naively apply this projection approach, the spread of the one-dimensional point sets could be much bigger than the spread R of the actual point set P (because the closest-pair distance of the one-dimensional sets could be much smaller).

To fix this problem, we extend our one-dimensional approach using the above lemma but in a more creative way. (Our idea can also extend the result of Feigenbaum et al. to higher dimensions, but our data structure is more efficient.) We always keep the location of the two newest points p_1 and p_2 in the current window. If $Q^{(\ell)} = \langle q_1, q_2, \dots, q_k \rangle$ is a summary sequence of projection of the given point set P on line ℓ , then before line 2 of the refine algorithm, we remove all q_i 's that satisfies $\|q_1 - q_i\| \leq \epsilon\|p_1 - p_2\|$ (after the points are projected). These points are too close to q_1 , and q_1 can serve as their representative. Condition 3 of summary sequences is still satisfied. Let $\delta(P)$ be the closest-pair distance of P . After the refine algorithm, $|Q|$ would then be bounded by $2 \log_{1+\epsilon} \frac{\Delta(P)}{\epsilon\delta(P)} = O(\frac{1}{\epsilon} \log \frac{R}{\epsilon})$.

Theorem 2. Let P be a point set in \mathbb{R}^d . Let \mathcal{B} be a set of $(1 + \epsilon)$ balls of radius Δ centered at points in P . Let \mathcal{B}' be a set of $O((\frac{1}{\epsilon})^{(d+1)/2} \log \frac{R}{\epsilon})$ balls of radius Δ centered at points in P . Let \mathcal{B}'' be a set of $O((\frac{1}{\epsilon})^{(d-1)/2})$ balls of radius Δ centered at points in P . Let \mathcal{B}''' be a set of $O((\frac{1}{\epsilon})^{(d-1)/2})$ balls of radius Δ centered at points in P .

Proof: Each of the $O((\frac{1}{\epsilon})^{(d-1)/2})$ one-dimensional structures stores $O(\frac{1}{\epsilon} \log \frac{R}{\epsilon})$ points. An approximate diameter can be computed by Lemma 3. An insertion/deletion of a point may cause insertions/deletions in any of these one-dimensional structures. \square

In the above theorem, we can also redefine R to be the ratio between the diameter and the minimum distance over all \dots pairs (instead of all arbitrary pairs).

5 Diameter in Higher Non-constant Dimensions

Although our algorithm can work in any fixed dimension, the complexity grows exponentially if d is not constant. If we allow a larger approximation factor,

more precisely, $\sqrt{d}(1 + \epsilon)$, we can still apply the projection approach to each of d main axes and get a structure that stores $O(d \log R)$ points for a fixed constant $\epsilon > 0$. To get an approximation factor independent of d , we suggest a different approach that requires larger (though sublinear) space. This approach in fact works for any metric space, where the distance function $d(\cdot, \cdot)$ satisfies the triangle inequality.

Lemma 4. For any metric space (X, d) , any $\alpha > 0$, and any points $o, o', p, q \in X$, $d(o, p) \leq \alpha d(o, q)$, and any $q' \in \{o, q\}$, we have $d(o', p) \leq (2\alpha + 1)d(o', q')$.

Proof:

$$\begin{aligned} d(o', p) &\leq d(o', o) + d(o, p) \leq d(o, o') + \alpha d(o, q) \\ &\leq d(o, o') + \alpha[d(o, o') + d(o', q)] \leq (2\alpha + 1) \max\{d(o', o), d(o', q)\}. \quad \square \end{aligned}$$

For a sequence of points C in a metric space, let o_C be the newest point in C and let δ_C be the distance of the closest point in C to o_C . Define Q_C , the summary set of C , as follows: For each $i \geq 0$, put the newest point among points at distance between $(1 + \epsilon)^i \delta_C$ and $(1 + \epsilon)^{i+1} \delta_C$ into Q_C ; also put o_C into Q_C .

Our algorithm proceeds as follows. We keep new points as they enter, and after every k insertions (for a parameter k to be determined later), we let C to be the set of k newest points (which we call a cluster) and replace C with Q_C . The main property regarding the summary set is that each point $p \in C$ has a representative $q \in Q_C$, not older than p , such that $\frac{1}{1+\epsilon}d(o_C, q) \leq d(o_C, p) \leq (1 + \epsilon)d(o_C, q)$. Since $|Q_C| = O(\log_{1+\epsilon} R)$ for each C , our data structure keeps $O(k + \frac{n}{k} \log_{1+\epsilon} R)$ points.

To approximate the diameter of the whole window, we consider the center o_C of the newest cluster C and find the furthest point from o_C among all of the points kept in our data structure. (Note that the furthest point from o_C can be updated in $O(1)$ time after an insertion.) By the main property of summary sets and Lemma 4, if p is the furthest point from o_C in the whole window, then we have kept a point q not older than p such that $d(o_C, p) \leq (2(1 + \epsilon) + 1)d(o_C, q)$. Since $d(o_C, p)$ is a 2-approximation of the diameter, we have a $6 + O(\epsilon)$ approximation of the diameter.

Deletion of the oldest point is easy. Setting $k = \sqrt{N}$ yields the following result:

Theorem 3. For any metric space (X, d) , any $\epsilon > 0$, and any sequence of points p_1, p_2, \dots, p_N in X , we can maintain a summary set S of size $O(\sqrt{N} \log R)$ such that for any point p_i in the window, there is a point $q \in S$ such that $d(p_i, q) \leq (6 + \epsilon)d(p_i, o_S)$. The update and deletion of S can be done in $O(1)$ time per operation. The space complexity is $O(\log R)$.

The above method is similar to Feigenbaum et al.'s one-dimensional method (Section 2), but instead of $O(\log N)$ levels of merging, we use only one level. In fact, if we use multiple levels of merging, we can get better space complexity at the expense of a bigger approximation factor. More precisely,

for any constant m , if we build a cluster for every $N^{\frac{1}{m+1}}$ points and merge the summary sets of every $N^{\frac{1}{m+1}}$ clusters of the same size together, then we can obtain a $(2^{m+2} - 2 + \epsilon)$ -approximation algorithm by storing $O(N^{\frac{1}{m+1}} \log R)$ points.

6 Two-Dimensional Core-Sets

In this section, we consider geometric problems more difficult than the diameter under the sliding window model. Specifically, we show how to maintain an ϵ -approximation of a two-dimensional point set, as defined by Agarwal et al. [1]:

Definition 2. Let $P \subseteq \mathbb{R}^d$ be a point set and $\omega(P, x)$ be the extent measure of P centered at $x \in \mathbb{R}^d$. A set $S \subseteq P$ is called an ϵ -core-set of P if $\omega(S, x) \geq \frac{1}{1+\epsilon} \omega(P, x)$ for all $x \in \mathbb{R}^d$.

Clearly, if S is an ϵ -core-set of a set P , then the width of S is a $(1 + \epsilon)$ -approximation of the width of P . Agarwal et al. [1] and Chan [7] gave efficient algorithms for constructing core-sets and applied them to develop approximation algorithms for various geometric optimization problems in both the traditional and data stream model.

Our algorithm in this section is for a fixed window size, and unfortunately its space bound depends on a new parameter in addition to ϵ , N , and R . This new parameter is the ratio between the diameter and the smallest width of each consecutive triple of points in the window and is denoted by R' . In Section 7 we show that the dependence on this new parameter is necessary for any algorithm that maintains an approximate width over a sliding window.

Our algorithm is a nontrivial combination of the diameter methods of Sections 2–4 with Chan’s core-set method for data streams [7]. We start by using the clustering approach of Section 2 as in Feigenbaum et al. [10], where we store $O(\log N)$ clusters, and in each cluster C we attempt to keep a small subset of points N_C as the representative points of C .

Maintaining D_C . For each cluster C , we first maintain a set D_C of candidate points for a constant-factor approximation of the farthest point in C to the center o_C . This is done by following the diameter structure of Section 4, with say $\epsilon = 1/2$ (which maintains a constant number of summary sequences). Whenever two clusters C_1 and C_2 are combined to form C , we can compute D_C from D_{C_1} and D_{C_2} by combining the corresponding summary sequences of D_{C_1} and D_{C_2} . More precisely to combine two summary sequences both on the same line, we put the two sequences together and run the refine algorithm. When a point is deleted from C , we simply delete it from D_C .

Lemma 5. Let $p, q \in D_C$ and o_C be the center of C . Then $\|o_C - p\| \leq 3\|o_C - q\|$.

Proof: Let q and r be the two furthest points in D_C ; we know $(1 + \epsilon)||q - r|| \geq \Delta_p(C)$, where $\Delta_p(C)$ denotes the diameter of points in C that are not older than p . Then

$$||o_C - p|| \leq \Delta_p(C) \leq (1 + \epsilon)||q - r|| \leq 2(1 + \epsilon) \max\{||o_C - q||, ||o_C - r||\}.$$

The lemma follows as we have chosen $\epsilon = \frac{1}{2}$. □

For each $q \in D_C$, consider $\frac{6}{\alpha}$ lines perpendicular to $\overrightarrow{o_C q}$ at distances $0, \alpha||o_C - q||, 2\alpha||o_C - q||, \dots, 3||o_C - q||$ from o_C , and call this set of lines L_q (the value of α will be specified later). For a point p and line ℓ , let $d(p, \ell)$ be the distance between p and ℓ . For $\ell \in L_q$, let C_ℓ be the set of points $p \in C$ such that $||o_C - p|| \leq 3||o_C - q||$ and ℓ is the closest line to p in L_q . Let o be the newest point in C_ℓ . Let o_C, s_1 , and s_2 be the newest, second newest, and third newest points C respectively and let w_C be the width of these three points.

Fix a point $q \in D_C$ and a line $\ell \in L_q$. Let p' denote the projection of a point p to ℓ and $C'_\ell = \{p' \mid p \in C_\ell\}$. For each $i \geq 0$, among the set of points whose projections are at distance between $(1 + \alpha)^i \alpha w_C$ and $(1 + \alpha)^{i+1} \alpha w_C$ from o' , choose the newest point as the representative of these points. Also for points whose projection are at distance at most αw_C from o' , choose o as their representative. With this selection, each point $p \in C_\ell$ has a representative v such that either $||v' - p'|| \leq \alpha w_C$ or

$$\frac{1}{1 + \alpha} \max\{||o' - v'||, ||o' - p' ||\} \leq ||o' - v' || \leq (1 + \alpha) \min\{||o' - v' ||, ||o' - p' ||\}. \tag{2}$$

We let $v_q(p)$ denote such a representative v . Notice that here each point may have several representatives (possibly a different one for each $q \in D_C$). Notice that if $||p - o_C|| > 3||q - o_C||$, then $v_q(p)$ does not exist.

The above approach and the proof of the following lemma are similar to Chan's approach for data streams [7].

Lemma 6. For any point $p \in C$ and any point $q \in D_C$, $|(v_q(p) - p) \cdot x| \leq O(\alpha)\omega_p(C, x)$.
 $\omega_p(C, x) = \max_{a,b}(a - b) \cdot x$ where $a, b \in C$ and a, b are not older than p .

Proof: By Lemma 5, pick a point $q \in D_C$ that is not older than p and $||o_C - p|| \leq 3||o_C - q||$. Let $v = v_q(p)$ and o be the center of the set C_ℓ that contains p . Then for any $a \in C_\ell$,

$$|(a - a') \cdot x| \leq \frac{\alpha}{2}(o - q) \cdot x \leq \frac{\alpha}{2}\omega_p(C, x), \tag{3}$$

since neither q nor o is older than p . Applying this to p and v , we have

$$|(v - p) \cdot x| \leq |(v - v') \cdot x| + |(v' - p') \cdot x| + |(p' - p) \cdot x| \leq |(v' - p') \cdot x| + \alpha\omega_p(C, x).$$

If $\|v' - p'\| \leq \alpha w_C$, then we are done. Otherwise, by Inequality 2, $\frac{1}{1+\alpha}\|o' - v'\| \leq \|o' - p'\| \leq (1 + \alpha)\|o' - v'\|$, implying that $|(v' - p') \cdot x| \leq \alpha|(o' - p') \cdot x|$. Then

$$\begin{aligned} |(o' - p') \cdot x| &\leq |(o' - o) \cdot x| + |(o - p) \cdot x| + |(p - p') \cdot x| \leq (1 + \alpha)\omega_p(C, x) \\ &\implies |(v - p) \cdot x| \leq (\alpha(1 + \alpha) + \alpha)\omega_p(C, x). \quad \square \end{aligned}$$

Lemma 3. N_C Whenever two clusters C_1 and C_2 are combined to form C , we construct N_C to be the set of all representatives of $N_{C_1} \cup N_{C_2}$. When a point is deleted from C , we simply delete it from N_C . By Lemma 6, each merging may cause an additive error of $O(\alpha)\omega_p(C, x)$. After a point p has experienced k merges, the additive error is $O(k\alpha)\omega_p(C, x)$. Since there could be $\log N$ merges, the final error will be $O(\alpha \log N)$, so to get an ϵ -core-set, we set $\alpha = \frac{\epsilon}{\log N}$.

For the space requirement of our method, observe that $|D_C| = O(\log R)$, as in Theorem 2. For each $q \in D_C$, there are $O(\frac{1}{\alpha})$ lines, each of which has $O(\log_{1+\alpha} \frac{R'}{\alpha})$ representatives. So, $|N_C| = O(\frac{1}{\alpha} \log R \log_{1+\alpha} \frac{R'}{\alpha}) = O(\frac{1}{\alpha^2} \log R \log \frac{R'}{\alpha})$. Since $\alpha = \frac{\epsilon}{\log N}$, the number of points in each cluster is $O(\frac{1}{\epsilon^2} \log^2 N \log R \log \frac{R' \log N}{\epsilon})$. As there are $\log N$ clusters, we obtain the following theorem:

Theorem 4. For any $\epsilon > 0$, there exists a core-set of size $O(\frac{1}{\epsilon^2} \log^3 N \log R (\log R' + \log \log N + \log \frac{1}{\epsilon}))$, such that the error is at most ϵ .

Using core-sets, approximation algorithms can be developed for many geometric optimization problems, including the k -median problem and the k -means problem [1].

7 A Lower Bound for Width

We now prove a space lower bound for any algorithm that maintains an approximate width in the sliding window model. Our goal is to support Theorem 4 by showing that the dependence of the space lower bound on R' is necessary.

We use a proof technique similar to Feigenbaum et al.'s [10].

Theorem 5. For any $\delta > 0$, there exists a core-set of size $R' \leq 2^{O(N^{1-\delta})}$ such that the error is at most δ .

Proof: Pick a non-increasing sequence $a_1 \geq a_2 \geq \dots \geq a_N$ from the set $\{c^{-3}, c^{-6}, \dots, c^{-3m}\}$ with $m = \frac{1}{3} \log_c R'$. Set $a_{N+1} = a_{N+2} = \dots = 1/R'$. Consider

the input stream of points $(-5 - i/N, 0)$, $(i/N, a_i)$, $(5 + i/N, 0)$ for $i = 1, 2, \dots, 2N$. Since the diameter is $O(1)$ and the closest-pair distance is $\Omega(1/N)$, the spread is $O(N)$. The width of any consecutive triple is at least $1/R'$. If at any time we have a c -approximation to the width, we can reconstruct a_1, \dots, a_N after the point $(1, a_N)$ is read. Let M be the number of non-increasing sequences of size N chosen from a set of size m . The number of bits stored by the algorithm must therefore be at least

$$\log M = \log \binom{N + m - 1}{m - 1} = \Omega \left(m \log \frac{N}{m} \right) = \Omega (\log R' \log N).$$

□

In contrast, if points have integer coordinates in the range $\{1, \dots, R\}$, it is possible to replace the parameter R' with R in Theorem 4, because the width of any non-collinear triple of points is lower-bounded by $\Omega(1/R)$.

References

1. P. K. Agarwal, S. Har-Peled, and R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, to appear.
2. P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1:189–201, 1991.
3. P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete & Computational Geometry*, 16:317–337, 1996.
4. G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38:91–109, 2001.
5. J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformations. *Journal of Algorithms*, 1(4):301–358, 1980.
6. T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal on Computational Geometry and Applications*, 12:67–85, 2002.
7. T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 152–159, 2004.
8. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(1):387–421, 1989.
9. C. A. Duncan, M. T. Goodrich, and E. A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 121–130, 1997.
10. J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, to appear; or as Tech. Report DCS/TR-1245, Yale University, <http://cs-www.cs.yale.edu/homes/jf/FKZ.ps>, 2002.
11. A. Goel, P. Indyk, and K. Varadarajan. Reductions among high dimensional proximity problems. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 769–778, 2001.
12. M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical Report SRC-TN-1998-011, Hewlett Packard Laboratories, 1998.

13. J. Hershberger and S. Suri. Convex hulls and related problems in data streams. In *ACM SIGMOD/PODS Workshop on Management and Processing of Data Streams*, pages 148–168, 2003.
14. P. Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 539–545, 2003.
15. S.M. Muthukrishnan. Datastreams: Algorithms and applications. Rutgers University Technical Report, <http://athos.rutgers.edu/~muthu/stream-1-1.ps>, 2003.
16. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
17. E. A. Ramos. An optimal deterministic algorithm for computing the diameter of a three-dimensional point set. *Discrete & Computational Geometry*, 26:233–244, 2001.

On-Line Windows Scheduling of Temporary Items

Wun-Tat Chan^{1,2,*} and Prudence W.H. Wong²

¹ Department of Computer Science, University of Hong Kong, Hong Kong
wtchan@cs.hku.hk

² Department of Computer Science, University of Liverpool, UK
pwong@csc.liv.ac.uk

Abstract. In this paper we study on-line windows scheduling (WS) of temporary items. In a broadcasting system, there are some broadcast channels, each can broadcast one item in one time unit. Upon the arrival of an item with a window w , where w is a positive integer, the item has to be scheduled on a channel such that it will be broadcasted in the channel at least once every w time units until its departure. With the on-line input of these temporary items, the problem is to minimize the maximum number of channels used over all time. We give a 5-competitive on-line algorithm and show that there is a lower bound of $2 - \epsilon$ for any $\epsilon > 0$ on the competitive ratio of any on-line algorithm.

1 Introduction

In this paper we study on-line windows scheduling (WS) [3, 4] of temporary items (i.e., items may depart). We are the first to consider temporary items in the problem. A temporary item may have *unknown departure time*, i.e., the item departs at unpredictable time; or *known departure time*, i.e., the departure time is known when the item arrives. In a broadcasting system, there are some broadcast channels, each can broadcast one item in one time unit. An item i comes with a window w_i , where w_i is a positive integer. When item i arrives, it has to be scheduled on one of the channels such that it will be broadcasted at least once every w_i time units. Reschedule of the item to other channels is not allowed. For example, suppose two items with windows equals to 2 and 3, respectively, arrive one after the other. Then we can schedule them in the same channel with the broadcast sequence $\langle 2, 3 \rangle$ or $\langle 2, 2, 3 \rangle$ repeatedly. The objective of the problem is to minimize the maximum number of channels used over all time.

WS can be applied to push-based broadcasting systems. In a push-based system, servers broadcast pages on broadcast channels to clients. Clients who wish to receive the pages will listen to the channels and select what information they want to receive. If the quality of service is measured by the average response time [1, 2], servers broadcast more popular pages more frequently. Yet, in some

* This research was supported in part by Hong Kong RGC Grant HKU-5172/03E.

business model, the servers sell their service to information providers who request the pages to be broadcasted regularly [5, 11]. The frequency of broadcasting a page is proportional to the amount paid by the provider. The servers may receive additional requests by providers over time, while a provider may withdraw from the service at any time; this is modeled by temporary items. Another related application is video-on-demand systems: many requests of a popular movie are received over a short period of time. To reduce the server bandwidth requirement without sacrificing the response time, the pyramid broadcasting paradigm [14, 15] and its variants [12, 13] are adopted. The movie is partitioned into equal size segments, each can be broadcasted in one time unit, and segment i is broadcasted every i time units. Then any client can view the movie uninterruptedly by waiting for at most one time unit.

Related Work: WS with permanent items (i.e., the items never depart) was first studied by Bar-Noy and Ladner [3], where they gave an off-line approximation algorithm which uses $H + O(\ln H)$ channels, for $H = \sum_i 1/w_i$. The NP-hardness of the problem was proved later [4]. An on-line algorithm was also given which uses $H + O(\sqrt{H})$ channels [4].

As pointed out in [4], WS is a restricted version of a bin packing problem, called the unit fraction bin packing (UFBP). UFBP is to pack all items, each with a width $1/w_i$ for some positive integer w_i , into minimum number of unit-capacity bins.

It can be observed in the following that WS is a restricted version of UFBP. Suppose we have three items with w_i equal to 2, 3, and 6. They can be packed in the same bin because $1/2 + 1/3 + 1/6 = 1$. Yet we cannot schedule to broadcast them in the same channel [8]. It has been shown in the pinwheel problem¹ [7, 8, 10] that if the sum of $1/w_i$ is less than or equal to $3/4$, the items can be scheduled on one channel [10], otherwise, it might not be possible to do so. Bar-Noy et al. [4] gave an off-line algorithm for UFBP that uses $H + 1$ bins, where $H = \sum_i 1/w_i$. They also gave an on-line algorithm that uses $H + O(\sqrt{H})$ bins and a lower bound of $H + \Omega(\ln H)$.

Coffman et al. [9] has studied the unit-capacity bin packing problem with temporary items, where the item width can be any real number less than or equal to 1. When the largest item width is 1, they showed that first-fit is 2.897-competitive and no on-line algorithm is better than 2.5-competitive. When the largest item width is less than $1/k$ for an integer k , they gave a general upper bound in terms of k for first-fit and a general lower bound in terms of k for any on-line algorithm. Note that the upper bound of UFBP with temporary items follows from the upper bound results of Coffman et al. but the lower bound does not.

Our Contribution: Similar to other study on on-line problems, we measure the performance of an on-line algorithm in terms of competitive ratio (see [6] for a survey). In this paper we give the first upper and lower bounds on the competitive ratio of WS with temporary items. We observe that the competitive

¹ The pinwheel problem is to determine whether the given items can be scheduled in one single channel and give the schedule if the answer is yes.

ratios depend on a value α which is the minimum of w_i . Precisely, we show that any on-line algorithm has a competitive ratio at least $1 + 1/\alpha - \epsilon$ for any $\epsilon > 0$. In contrast to the case of permanent items in which there is an algorithm that uses at most $H + O(\sqrt{H})$ channels, the $1 + 1/\alpha - \epsilon$ lower bound demonstrates that WS with temporary items is “harder” than WS with permanent items. This lower bound applies to both known and unknown duration, yet the adversary for unknown duration is simpler as expected. As for upper bound, we give an on-line algorithm \mathcal{W} that is 5-competitive when $\alpha = 1$, and $(2 + 2/(\alpha^* - 1))$ -competitive when $\alpha \geq 2$, where α^* is the largest power of 2 that is smaller than or equal to α . These upper bounds also hold for both known and unknown duration.

Organization of the Paper: The rest of the paper is organized as follows. In Section 2, we give some preliminaries. In Section 3, we present the lower bound for UFBP and WS. The lower bound for WS is based on the lower bound for UFBP. In Section 4, we present the on-line algorithm \mathcal{W} for WS and analyze its performance. Finally, we give a conclusion in Section 5.

2 Preliminaries

In this section, we give a precise definition of the on-line problems of UFBP and WS with temporary items, and the necessary notations for further discussion.

On-Line UFBP: The input is a sequence σ of items. Item i is represented as a 3-tuple $(a_i, d_i, 1/w_i)$, where a_i , d_i , and w_i are positive integers with a_i denoting the arrival time, d_i the departure time, and $1/w_i$ the width of item i . For unknown duration, d_i is not specified when item i arrives. All items are to be packed in unit-capacity bins. The objective of the problem is to minimize the maximum number of bins used over all time.

On-Line WS: The input is a sequence σ of items for broadcast. Item i is represented as a 3-tuple (a_i, d_i, w_i) , where a_i , d_i , and w_i are positive integers denoting the arrival time, the departure time, and the window of item i , respectively. We assume that $d_i \geq a_i + w_i - 1$, i.e., an item that arrives should be broadcasted at least once. For unknown duration, d_i is not specified when item i arrives. Each item takes one unit of time to broadcast, i.e., if an item is broadcasted in the beginning of time t , the client can receive the item at the end of time t . Item i must be scheduled on a broadcast channel between the time interval $[a_i, d_i]$ such that for any $a_i \leq t_1 \leq t_2 \leq d_i$ and $t_2 - t_1 + 1 \geq w_i$, item i should be broadcasted at least once in the sub-interval $[t_1, t_2]$. Note that in this definition, the first broadcast of item i must be within the time interval $[a_i, a_i + w_i - 1]$, i.e., there are at most $w_i - 1$ time units allowed between the arrival time and the first broadcast of an item. Similarly, we assume that in the unknown duration case, item i should announce its departure at or before time $d_i - w_i + 1$. The objective of the problem is to minimize the maximum number of channels used over all time. Define the *load* of a channel at time t to be the sum of $1/w_i$ for all item i scheduled in the channel that have arrived but not yet departed.

Both the above problems are on-line problems in the sense that at any time t , there is no information on the items arriving after t . Given a WS (UFBP, resp.) algorithm \mathcal{W} , we analyze its performance using competitive analysis. Given a sequence σ of items, let $\mathcal{W}(\sigma, t)$ be the number of channels (bins, resp.) used by \mathcal{W} at time t . We say that \mathcal{W} is c -competitive if there exists a constant k such that for any input sequence σ , we have

$$\max_t \mathcal{W}(\sigma, t) \leq c \cdot \max_t \mathcal{O}(\sigma, t) + k,$$

where \mathcal{O} is the optimal off-line algorithm of WS (UFBP, resp.).

Remarks: Notice that any on-line algorithm for WS with unknown duration always maintains the load of every channel to be at most 1 because it has to guarantee that all items scheduled on a channel can be broadcasted within their windows (even if none of the items depart). Therefore, we also restrict our attention to those off-line algorithms that maintain, at any time, the load of every channel to be at most 1.

3 Lower Bound Results

In this section we present lower bounds for UFBP and WS with temporary items. The lower bound for UFBP is easier to construct, based on which we can construct the lower bound for WS. Precisely, we show that for both UFBP and WS, no on-line algorithm is better than $(1 + 1/\alpha - \epsilon)$ -competitive, for any $\epsilon > 0$ and $\alpha = \min_i \{w_i\}$, this is true for both known and unknown duration. Yet the adversary for known duration is more complicated as expected and is based on the adversary for unknown duration. We will give the details of the lower bound for UFBP, the lower bound for WS can be proved similarly and will be outlined at the end of this section.

Lower Bound for UFBP. We first consider the case of unknown duration. Given any on-line algorithm \mathcal{A} , we construct a sequence of items of widths either $1/y$ or $1/\alpha$ for some y and α such that y is much greater than α . The adversary works in three stages as follows.

1. Items with $w_i = y$ are released at time $a_i = i$ until at least y bins are used by \mathcal{A} and the number of items m released is a multiple of y . Consider the minimum such m .
2. The adversary retains one item in each of any y occupied bins and let all the other $m - y$ items depart.
3. Finally, $\alpha(m - y)/y$ items with $w_i = \alpha$ are released.

Notice that at most y^2 items with $w_i = y$ are sufficient to force the on-line algorithm to use at least y bins in Stage 1. Thus, $m \leq y^2$. The following lemma gives a lower bound on the number of bins used by the on-line algorithm after Stage 3.

Lemma 1. \mathcal{A} $y + \max\{0, (m - y)/y - y(\alpha - 1)/\alpha\}$. . .

Notice that after Stage 2, at most $\alpha - 1$ items of width $1/\alpha$ can be packed in each of the y occupied bins. The total width of the items released in Stage 3 is equal to $(m - y)/y$. Therefore, if $(m - y)/y > y(\alpha - 1)/\alpha$, \mathcal{A} needs at least $(m - y)/y - y(\alpha - 1)/\alpha$ additional new bins to pack all items released in Stage 3. Hence, the number of bins used is at least $y + \max\{0, (m - y)/y - y(\alpha - 1)/\alpha\}$, and the lemma follows. \square

The following theorem shows the lower bound on the competitive ratio for UFBP with unknown duration. Roughly speaking, we establish the lower bound by showing that there is an off-line algorithm that uses m/y bins at any time and then compare the number of bins used by the on-line algorithm with m/y .

Theorem 1. *For any $\epsilon > 0$ and $\alpha = \min_i\{w_i\}$, there exists an off-line algorithm that uses at most $(1 + 1/\alpha - \epsilon)m/y$ bins at any time.*

First we show that there is an off-line algorithm \mathcal{O} that uses m/y bins at any time. In Stage 1, \mathcal{O} packs the y non-departing items (i.e., those remain after Stage 2) in the same bin and the other $m - y$ items in another $(m - y)/y$ bins. In Stage 3, $\alpha(m - y)/y$ items of width $1/\alpha$ are released. These items are packed by \mathcal{O} into the $(m - y)/y$ bins that become empty after Stage 2. Thus, the number of bins used by \mathcal{O} at any time is $1 + (m - y)/y = m/y$.

Then, we analyze the number of bins used by any on-line algorithm \mathcal{A} . There are two cases to consider. (1) Suppose that $(m - y)/y \leq y(\alpha - 1)/\alpha$. By Lemma 1, no new bins are needed in Stage 3 and the maximum number of bins used by \mathcal{A} at any time is y . By simple mathematics, the inequality can be rearranged as $y^2/m \geq 1/(1 + 1/y - 1/\alpha)$. Therefore, the ratio of the maximum number of bins used by \mathcal{A} to that used by \mathcal{O} is $y/(m/y)$, which is at least $1/(1 + 1/y - 1/\alpha) \geq 1 + 1/\alpha$ for $y \geq (\alpha + 1)/(\alpha + 1/\alpha - 1)$. (2) Suppose that $(m - y)/y > y(\alpha - 1)/\alpha$. The maximum number of bins used by \mathcal{A} at any time is $y + (m - y)/y - y(\alpha - 1)/\alpha = m/y + y(1/\alpha - 1/y) \geq m/y + (m/y)(1/\alpha - 1/y)$. The latter inequality holds because $m \leq y^2$. Therefore, the ratio of the maximum number of bins used by \mathcal{A} to that used by \mathcal{O} is at least $1 + 1/\alpha - 1/y$. By letting $\epsilon = 1/y$, the ratios in both case are at least $1 + 1/\alpha - \epsilon$, thus, the theorem holds. \square

We then modify the adversary such that it also works for known duration. The major issue is that we have to determine the departure time of the items when they arrive such that at the end of Stage 2, the on-line algorithm uses y bins, each containing exactly one item of width $1/y$. Then, using a similar argument in Theorem 1, we can prove the same lower bound for known duration. Roughly speaking, the departure time of the item i , for $i \geq 2$, depends on how the on-line algorithm pack item $(i - 1)$, in particular, whether item $(i - 1)$ is packed in an empty bin or not. The details are as follows.

To set the departure time d_i of item i arriving at time i , we need to consider how the on-line algorithm \mathcal{A} packs the items arrived so far. We capture this information by two values β_{i-1} and γ_{i-1} (to be defined). For any time t , let I_t be the set of items arrived at or before t and $F_t \subseteq I_t$ be the set of items that \mathcal{A} opens an empty bin for. We define $\beta_t = \min_{i \in F_t} \{d_i\}$ and $\gamma_t = \max_{i \in I_t - F_t} \{d_i\}$.

Now, we describe how to set the departure times. Firstly, we set $d_1 = L$ for some large constant L to be defined later. Note that $\beta_1 = L$ and $\gamma_1 = 0$. Next, we set $d_2 = (\beta_1 + \gamma_1)/2 = L/2$. If \mathcal{A} packs item 2 in the same bin as item 1, we have $\beta_2 = L$ and $\gamma_2 = L/2$. Otherwise, we have $\beta_2 = L/2$ and $\gamma_2 = 0$. In general, we set $d_i = (\beta_{i-1} + \gamma_{i-1})/2$. Notice that Stage 2 ends at time γ_m and Stage 3 starts at time $\gamma_m + 1$. For the adversary to work, we need to ensure that d_i is an integer and $d_i > m$ for all $1 \leq i \leq m$, which can be proved to hold if $L \geq 2^{y+\lceil \log m+1 \rceil}$. The details will be given in the full paper.

The following lemma gives an invariant about the departure times.

Lemma 2. *For any t such that $1 \leq t \leq \gamma_m$, we have $\beta_t > \gamma_t$.*

We first consider the case where $1 \leq t \leq m$. We prove the lemma by a stronger claim that (1) if item t is packed in an empty bin by \mathcal{A} , then $\beta_t = d_t$, otherwise, $\gamma_t = d_t$; and (2) $\beta_t > \gamma_t$. We prove the claim by an induction on t . The claim holds for $t = 1$ because $d_1 = \beta_1 = L$ and $\gamma_1 = 0$. Assume the claim holds for some $t < k \leq m$. At time k , item k arrives. Since $d_k = (\beta_{k-1} + \gamma_{k-1})/2$ and $\beta_{k-1} > \gamma_{k-1}$, we have $\gamma_{k-1} < d_k < \beta_{k-1}$. If item k is packed in an empty bin, then $\beta_k = \min\{d_k, \beta_{k-1}\} = d_k$ and $\gamma_k = \gamma_{k-1}$. Otherwise, $\gamma_k = \max\{d_k, \gamma_{k-1}\} = d_k$ and $\beta_k = \beta_{k-1}$. For both cases, we have $\beta_k > \gamma_k$, thus, the claim holds.

For any $m < t \leq \gamma_m$, there is no item released at time t . Therefore, we have $\beta_t = \beta_m > \gamma_m = \gamma_t$. Combining with the above claim, the lemma follows. \square

Using the invariant of Lemma 2 and a similar argument as Theorem 1, we have the following theorem.

Theorem 2. *For any $\epsilon > 0$ and $\alpha = \min_i \{w_i\}$, there exists a constant c such that for any $y \geq c$, the competitive ratio of any on-line algorithm for UFBP with known duration is at least $(1 + 1/\alpha - \epsilon)$.*

Notice that with known duration, we can have an off-line algorithm using the same number of bins as the one given in the proof of Theorem 1. As a result, we only need to consider the number of bins used by any on-line algorithm. By Lemma 2, we can see that at time $\gamma_m + 1$ each of the y occupied bins has exactly one item of width $1/y$. By Lemma 1, the on-line algorithm uses at least $y + \max\{0, (m - y)/y - y(a - 1)/\alpha\}$ bins. Following the same argument as in Theorem 1, we can prove that every on-line algorithm for UFBP with known duration has competitive ratio at least $1 + 1/\alpha - \epsilon$ for any $\epsilon > 0$ and $\alpha = \min_i \{w_i\}$. \square

Lower Bound for WS. To prove the lower bound for WS, we modify the adversary for UFBP such that a bin is considered as a channel and an item of width $1/w$ becomes an item of window w . Then, we have an adversary for the on-line WS. Based on the modified adversary, we can derive the following theorem.

Theorem 3. *For any $\epsilon > 0$ and $\alpha = \min_i \{w_i\}$, there exists a constant c such that for any $y \geq c$, the competitive ratio of any on-line algorithm for WS with known duration is at least $(1 + 1/\alpha - \epsilon)$.*

4 Upper Bound Results

In this section, we give an on-line algorithm \mathcal{W} for the WS problem and analyze its performance. We focus on the unknown duration case, the result carries for the known duration case. Roughly speaking, the on-line algorithm \mathcal{W} rounds each item window w down to w' which is a power of two (e.g., we round the window of 7 down to 4) and then schedule the item according to w' . Note that if any item is broadcasted at least once in every interval of w' , it is broadcasted at least once in every interval of w . As a result, we can first focus on scheduling items with windows that are powers of two.

4.1 Broadcast Trees - Representation of Schedules

Similar to the work by Bar-Noy et al. [4], we represent a broadcast schedule on m channels by a forest of m binary trees. The schedule on each channel is represented by a binary tree in which all nodes have exactly zero or two children. We call this binary tree a broadcast tree. Given any broadcast tree, the schedule is to alternately broadcast an item from the left and right subtrees; items from each subtree T is selected alternately from the left and right subtrees of T in a recursive manner. For example, in Figure 1, the broadcast tree represents a schedule that alternates between the item a and the items on the right subtree; when selecting the right subtree, we alternate between the item b and items on the right subtree recursively. The corresponding schedule is $\langle abacabad \rangle$. Notice that a leaf at depth d represents an item scheduled with window $w = 2^d$.

To ease the discussion of the on-line algorithm \mathcal{W} , we give some definitions on broadcast trees. We say that a leaf is closed if there is no item assigned to the leaf, otherwise, it is open. An open leaf is represented by an unfilled circle and closed by filled circle. We label a leaf at depth d by 2^d . See Figure 2 (a) for an example. The value of a broadcast tree is defined to be the sum of reciprocal of the labels of all closed leaves in the tree. A lace binary tree of height h is a binary tree in which for each depth from 1 to $h - 1$, there is a single leaf and for depth h , there are two leaves. Note that there are more than one different lace binary tree of height $h > 2$. The tree in Figure 2 (c) is a lace binary tree of height 3.

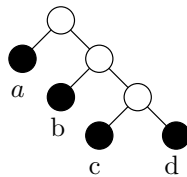


Fig. 1. The schedule corresponding to this broadcast tree is $\langle abacabad \rangle$

4.2 The On-Line Algorithm \mathcal{W}

The on-line algorithm by Bar-Noy et al. expands an existing broadcast tree or opens a new broadcast tree when items arrive. Our on-line algorithm \mathcal{W} handles

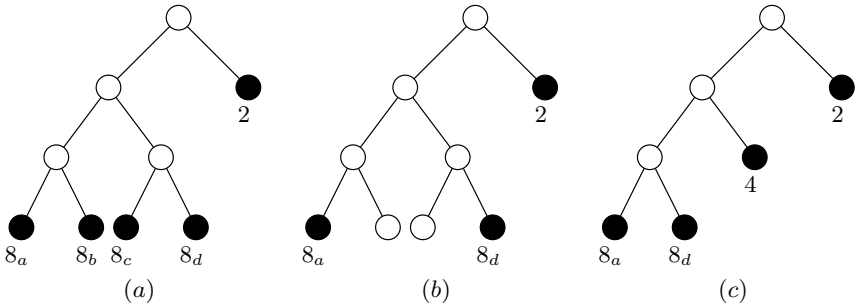


Fig. 2. Appropriate rearrangement is necessary after departure of items. (a) Broadcast tree for the sequence $\{8_a, 8_b, 8_c, 8_d, 2\}$. (b) Items 8_b and 8_c depart; a new item 4 cannot be included into the tree. (c) The item 4 can be included if the tree is restructured

newly arrived items similarly. Yet when items depart, we have to rearrange a broadcast tree wisely to make sure that new items can still be inserted to a broadcast tree as long as the tree is not very full. We illustrate the need for proper rearrangement in the following example. Suppose there are 4 items of window 8 and 1 item of window 2 arriving and the broadcast tree is as shown in Figure 2 (a). Later the second and third items of window 8 depart and another item of window 4 arrives. Figure 2 (b) shows the broadcast tree after the items depart if we do not rearrange the tree properly; we cannot include the item of window 4 into the tree. Yet if we rearrange the tree as in Figure 2 (c), we are able to include the new item into the tree.

Now, we describe the on-line algorithm \mathcal{W} . The on-line algorithm \mathcal{W} attempts to maintain an invariant on the structure of the broadcast trees: for every broadcast tree, there is at most one open leaf at each depth. To keep this invariant, the on-line algorithm \mathcal{W} modifies the structure according to the arrival and departure of items as follows.

Arrival: When an item with window w arrives, round it down to the nearest power of 2, say $w' = 2^v$. If there is an open leaf at depth v in some broadcast tree, schedule the item to that leaf. Otherwise, let $u < v$ be the maximum value such that there is an open leaf at depth u in some broadcast tree. If no such u exists, open a new tree with one open leaf (in this case $u = 0$). Let ℓ be the leaf selected. Append a lace subtree of height $v - u$ with all leaves open to the tree to replace ℓ and then schedule the new item to one of the open leaves at depth v in the resulting tree. See Figure 3 (a)-(d) for examples.

Departure: When an item at depth v of a broadcast tree is going to depart in the next 2^v time, the corresponding leaf ℓ will become open. If there is no other open leaf at depth v of the tree, no restructuring is needed. Otherwise, there is exactly one other open leaf f . Let ℓ' be the node, a closed leaf or an internal node, that shares the same parent p with ℓ in the tree. Detach the

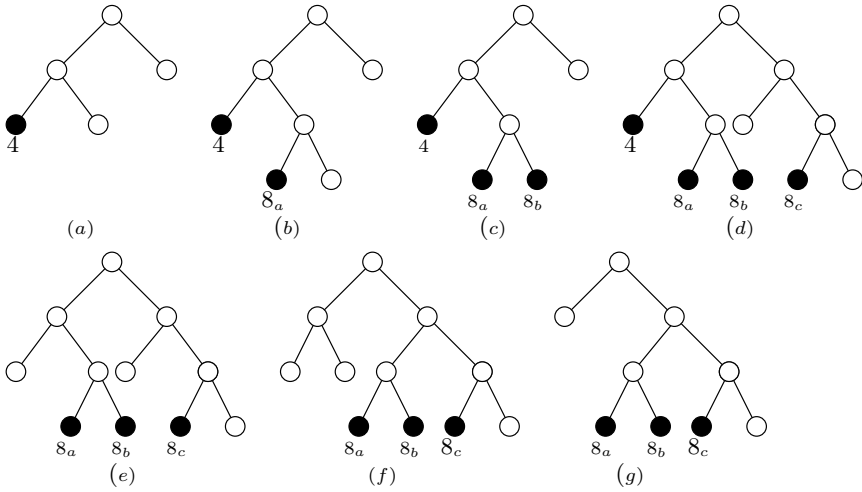


Fig. 3. (a)-(d) Evolution of the broadcast tree for the sequence $\{4, 8_a, 8_b, 8_c\}$. Note that the lace binary trees inserted are chosen for better drawing effect only. (e)-(g) Evolution of the broadcast tree when the item 4 departs

subtree rooted at ℓ' and replace f by this subtree. Remove ℓ and make p an open leaf. Then we repeat the restructuring similarly at depth $v - 1$ and upper depth, if necessary, by considering p as the new open leaf. See Figure 3 (e)-(f) for an example.

Remarks: When we transit from the old to the new broadcast schedule, the interval between the broadcast of an item that has been moved is altered. To make sure that the interval is still smaller than or equal to its window size, we will broadcast the item once more at the original time slot (which becomes an empty slot in the new schedule), if necessary. This can be proved that the remedy guarantees the broadcast interval to be at most the window size of the moved item, and the transition lasts for at most 2^v time units. The details will be given in the full paper.

4.3 Analysis of the On-Line Algorithm \mathcal{W}

In this section, we analyze the performance of the on-line algorithm \mathcal{W} . Roughly speaking, we show that when the on-line algorithm \mathcal{W} opens a new broadcast tree for an item, the load in each existing broadcast tree is reasonably close to 1. Since the optimal off-line algorithm can schedule in each channel items with load at most 1, we can show that the number of channels used by the on-line algorithm \mathcal{W} is at most a constant times that is used by the optimal off-line algorithm. The first step is to show in Lemmas 3 and 4 that a broadcast tree is reasonably full whenever we cannot schedule a new item in the tree.

Lemma 3.

The proof is by induction on time. Initially, there is only one open leaf at depth zero (the root of the first broadcast tree). When a new item arrives, we either close an open leaf or replace an open leaf ℓ with a lace tree. Note that ℓ is chosen in a way that its depth is the largest among all candidate open leaves. That means there is no open leaf on each depth corresponding to the added lace tree. Furthermore, the lace tree only contains one single leaf in each depth except the bottommost one in which one of the two open leaves is assigned to the new item. As a result, adding a new item to a broadcast tree keeps the invariant.

When an item departs, a closed leaf becomes open. If there is no other open leaf at the same depth, the number of open leaf is then one. If there is another open leaf at the same depth, the broadcast tree will be restructured so that the two open leaves are attached to the same parent. The two open leaves will be removed and then the number of open leaves will become zero. However, the parent of the two open leaves is made open, thus increase the number of open leaves at that depth. The restructuring procedure is repeated, if necessary, and thus keeping at most one open leaf at other depth. \square

Lemma 4. $\dots\dots\dots 2^v \dots\dots\dots$
 $\dots\dots\dots T \dots\dots\dots T \dots\dots\dots 1 - 1/2^v$

By the definition of the on-line algorithm \mathcal{W} , there is no open leaf at depth $v' \leq v$. By Lemma 3, there is at most one open leaf at depth greater than v . Notice that a broadcast tree with all leaves closed has a load of 1. Therefore, the total load of T is at least $1 - \sum_{u>v} 1/2^u \geq 1 - 1/2^v$. \square

Based on the above two lemmas, we can derive the competitive ratio of the on-line algorithm \mathcal{W} as follows.

Theorem 4. $\dots\dots\dots \mathcal{W} \dots\dots\dots 2 + 2/(\alpha^* - 1) \dots\dots\dots \alpha^* \geq 2, \dots\dots\dots \alpha^* \dots\dots\dots 2 \dots\dots\dots \min_i\{w_i\} \dots\dots\dots$

Suppose the on-line algorithm \mathcal{W} opens a new broadcast tree for a new item with window w and $w' = 2^v$ is the corresponding round down window. For any broadcast tree T in the forest, by Lemma 4, the total load of T is at least $1 - 1/2^v$. Thus, the total load of the corresponding channel is at least $(1 - 1/2^v)/2 = 1/2 - 1/2^{v+1}$ (because of the round down to nearest power of 2).

Let m be the number of channels used by the on-line algorithm \mathcal{W} . Then the total load of items, and hence the number of channels used by the optimal off-line algorithm is at least $(m - 1)(1/2 - 1/2^{v+1})$. Therefore, by taking an appropriate additive constant, the competitive ratio is at most $1/(1/2 - 1/2^{v+1}) = 2 + 2/(2^v - 1) \geq 2 + 2/(\alpha^* - 1)$. Thus, the theorem follows. \square

Theorem 5. $\dots\dots\dots \mathcal{W} \dots\dots\dots 5 \dots\dots\dots \min_i\{w_i\} = 1$

At any instance, suppose the on-line algorithm \mathcal{W} uses n channels for items with window 1 and m channels for items with other window values. Then

the optimal off-line algorithm must use at least n channels at this instance. Furthermore, consider the moment when the on-line algorithm \mathcal{W} opens the m -th channels for item with window $w > 1$, using a similar argument as in Theorem 4, the optimal off-line algorithm needs at least $(m - 1)/4$ channels (set $\alpha^* = 2$ in the formula in Theorem 4). In other words, the optimal off-line algorithm uses at least $\max\{n, (m-1)/4\}$ channels while the on-line algorithm \mathcal{W} uses $m + n$ channels. The worst case ratio is obtained when $n = (m - 1)/4$; by taking appropriate additive constant, the competitive ratio is at most 5 and the theorem follows. \square

5 Conclusion

In this paper we study on-line windows scheduling of temporary items. We have given a 5-competitive on-line algorithm and showed that there is a lower bound of $2 - \epsilon$ for any $\epsilon > 0$ on the competitive ratio of any on-line algorithm. An immediate open question is whether we can close the gap. Another interesting problem is to schedule as many items as possible when the number of channels is fixed; this involves admission control.

References

- [1] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *IEEE Personal Communications*, 2(6):50–60, 1995.
- [2] M. H. Ammar and J. W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5(4):235–242, 1985.
- [3] A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing*, 32(4):1091–1113, 2003. (A preliminary version appears in SODA 2002.).
- [4] A. Bar-Noy, R. E. Ladner, and T. Tamir. Windows scheduling as a restricted version of bin packing. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 224–233, 2004.
- [5] A. Bar-Noy, J. Naor, and B. Schieber. Pushing dependent data in clients-providers-servers systems. *Wireless Network*, 9(5):421–430, 2003.
- [6] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [7] M. Y. Chan and F. Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Transactions on Computers*, 41(6):755–768, 1992.
- [8] M. Y. Chan and F. Y. L. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–625, 1993.
- [9] E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 1983.
- [10] P. C. Fishburn and J. C. Lagarias. Pinwheel scheduling: Achievable densities. *Algorithmica*, 34(1):14–38, 2002.
- [11] V. Gondhalekar, R. Jain, and J. Werth. Scheduling on airdisks: Efficient access to personalized information services via periodic wireless data broadcast. In *Proceedings of IEEE International Conference on Communications*, volume 3, pages 1276–1280, 1997.

- [12] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of ACM SIGCOMM conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 89–100, 1997.
- [13] L.-S. Juhn and L.-M. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, 1997.
- [14] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video-on-demand service. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 66–77, 1995.
- [15] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *ACM Journal of Multimedia Systems*, 4(3):197–208, 1996.

Generalized Geometric Approaches for Leaf Sequencing Problems in Radiation Therapy*

Danny Z. Chen¹, Xiaobo S. Hu¹, Shuang Luan^{2,**}, Shahid A. Naqvi³,
Chao Wang¹, and Cedric X. Yu³

¹ Department of Computer Science and Engineering,
University of Notre Dame,
Notre Dame, IN 46556, USA
{chen, hu, cwang1}@cse.nd.edu

² Department of Computer Science,
University of New Mexico,
Albuquerque, NM 87131, USA
sluan@unm.edu.

³ Department of Radiation Oncology,
University of Maryland School of Medicine,
Baltimore, MD 21201-1595, USA
{snaqv001, cyu002}@umaryland.edu

Abstract. The 3-D *static leaf sequencing* (SLS) problem arises in radiation therapy for cancer treatments, aiming to deliver a prescribed radiation dose to a target tumor accurately and efficiently. The treatment time and machine delivery error are two crucial factors of a solution (i.e., a treatment plan) for the SLS problem. In this paper, we prove that the 3-D SLS problem is NP-hard, and present the first ever algorithm for the 3-D SLS problem that can determine a tradeoff between the treatment time and machine delivery error (also called the “tongue-and-groove” error in medical literature). Our new 3-D SLS algorithm with error control gives the users (e.g., physicians) the option of specifying a machine delivery error bound, and subject to the given error bound, the algorithm computes a treatment plan with the minimum treatment time. We formulate the SLS problem with error control as computing a k -weight shortest path in a directed graph and build the graph by computing g -matchings and minimum cost flows. Further, we extend our 3-D SLS algorithm to the popular radiotherapy machine models with different constraints. In our extensions, we model the SLS problems for some of the radiotherapy systems as computing a minimum g -path cover of a directed acyclic graph. We implemented our new 3-D SLS algorithm suite and conducted an extensive comparison study with commercial planning systems and well-known algorithms in medical literature. Some of our experimental results based on real medical data are presented.

* This research was supported in part by the National Science Foundation under Grant CCR-9988468.

** Corresponding author. The research of this author was supported in part by a faculty start-up fund from the Department of Computer Science, University of New Mexico.

1 Introduction

In this paper, we study the 3-D *step-and-shoot* (SLS) problem in *intensity modulated radiation therapy* (IMRT). IMRT is a modern cancer therapy technique and aims to deliver a high radiation dose that is as conformal to a tumor as possible. Performing IMRT is based on the ability to *step* and *shoot* and deliver prescribed discrete dose distributions of radiation, called *intensity maps* (IM). An IM is a dose prescription specified by a set of nonnegative integers on a 2-D grid (see Figure 1(a)). The value in each grid cell indicates the amount (in units) of radiation to be delivered to the body region corresponding to that IM cell.

One of the most advanced tools today for delivering intensity maps is the *multileaf collimator* (MLC) [11]. An MLC consists of up to 60 pairs of tungsten leaves (see Figure 1(b)). The leaves can move up and down to form a rectilinear region, called an *aperture*. To reduce radiation leakage, two pairs of backup metal diaphragms (along the x and y axes) are used to form a “bounding box” of each rectilinear MLC-aperture.

Currently, there are three popular MLC systems in clinical use. They are the Elekta, the Siemens, and the Varian MLC systems [11]. Depending on the specific MLC in use, there are some differences among the geometric shapes of the “deliverable” MLC-apertures. In this abstract, we will focus on the Elekta MLC system, and leave the details of Siemens and Varian MLCs to the full paper. (Section 2 describes the geometric properties of an Elekta MLC-aperture.)

There are several techniques for delivering IMRT [18, 19]. The most popular one is called the *step-and-shoot* (SLS) or the “*step-and-shoot*” technique [9, 19, 20]. In the SLS approach, an IM is delivered as follows: Form an MLC-aperture, turn on the beam source to deliver radiation to the area of the IM exposed by the MLC-aperture, turn off the beam source, reposition MLC leaves to form another MLC-aperture, and repeat until the entire IM is done. In this setting, the boundary of each MLC-aperture does not intersect the interior of any IM cell. In delivering a beam shaped by an MLC-aperture, all the cells inside the region of the MLC-aperture receive the same integral amount of radiation dose (say, one unit), i.e., the numbers in all such cells are decreased by the same integer value. The IM is done when each cell has a value zero.

A *delivery plan* for a given IM consists of a set of MLC-apertures for delivering the IM. (Each MLC-aperture is associated with an integral value that represent the amount of radiation prescribed to it.) Two key criteria are used to evaluate the quality of an IMRT treatment plan:

(1) **Treatment Time (the efficiency):** Minimizing the treatment time is important because it not only lowers the patients’ costs but also enables more patients to be treated. Since the overhead associated with turning the beam source on/off and repositioning MLC leaves dominates the total treatment time [9, 20], we like to minimize the number of MLC-apertures used for delivering an IM.

(2) **Machine Delivery Error (the accuracy):** Ideally, an IM is partitioned into a set of MLC-apertures that delivers the IM perfectly. However, in reality,

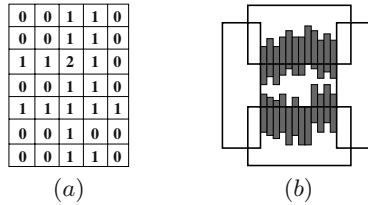


Fig. 1. (a) An IM. (b) Illustrating the Elekta MLC systems. The shaded rectangles represent the MLC leaves, and the unshaded rectangles represent the backup diaphragms

due to the special geometric shape of the MLC leaves [21], an MLC-aperture cannot be delivered as perfectly as its geometry. This gives rise to an error between the prescribed dose and actually delivered dose. We call this error the *machine delivery error*, which is also called the *leaf transmission error* in medical literature [17, 20, 21] (Section 2 discusses more on its nature). Minimizing the machine delivery error is important because the maximum machine delivery error can be up to 10% [10], which is well beyond the allowed 3–5% limit. Carefully choosing MLC-apertures to use can reduce the error.

The 3-D Static Leaf Sequencing (SLS) Problem is: Given an IM and an error bound \mathcal{E} , find a set S of MLC-apertures for delivering the IM such that the total machine delivery error incurred by S is $\leq \mathcal{E}$, and the size $|S|$ is minimized. Note that the key to the problem is to find a good tradeoff between the accuracy (error) and efficiency (treatment time).

The 3-D SLS problem has received a great deal of attention from medical physics community [4, 5, 6, 9, 17, 20], computational geometry [7, 8, 15], and operations research [1, 3, 13]. An influential paper is by Xia and Verhey [20] for solving the 3-D SLS problem **without** error control (i.e., the machine delivery error is ignored). Their algorithm has been implemented by many researchers and used as a widely accepted benchmark program for leaf sequencing software. However, none of the known approaches so far can solve the 3-D SLS problem **with** machine delivery error control (i.e., determining a good tradeoff between the error and time).

The main results of this paper are summarized as follows:

- (1) We prove that the 3-D SLS problem is NP-hard under all three popular MLC models, and with or without machine delivery error control.
- (2) We present the first ever modeling of the 3-D SLS problem with a tradeoff between the error and treatment time, and an efficient algorithm for the problem on the Elekta model. In our solution, the 3-D SLS problem is formulated as a k -weight shortest path problem on a directed graph, in which each edge is defined by a minimum weight g -cardinality matching. Every such k -weight path specifies a set S of k MLC-apertures for delivering the given IM, and the cost of the path indicates the machine delivery error of the set S of MLC-apertures.
- (3) We extend our 3-D SLS algorithm to other MLC models, such as Siemens and Varian. Our extensions are based on computing a minimum g -path cover of a directed acyclic graph.

(4) We also solve the 2-D case of the 3-D SLS problem in which the given IM consists of entries of only 1 or 0. Our solution is based on computing a minimum cost flow in a special graph.

(5) We implemented our algorithm suite and carried out experimental studies using medical data.

Due to space limit, we will focus on results (2) and (5) in this extended abstract, and leave the details of results (1), (3) and (4) to the full paper.

2 Preliminaries and Problem Statement

2.1 Constraints of Multileaf Collimators and Their Geometry

The mechanical constraints of an MLC preclude certain aperture shapes from being used [11]. One such constraint is called the *leaf separation constraint*, which requires the distance between the opposite leaves of any MLC leaf pair to be \geq a given value δ (say, $\delta = 1\text{cm}$). Another constraint is called the *no-interleaf motion constraint*, which forbids the tip of each MLC leaf to surpass those of its neighboring leaves on the opposite leaf bank. These constraints prevent opposite MLC leaves from colliding into each other and being damaged. The Elekta MLC (i.e., the default MLC system in this paper) is subject to both the leaf separation and the no-interleaf motion constraint, hence geometrically, each Elekta MLC-aperture is a rectilinear x -monotone *simple* polygon whose minimum vertical “width” is \geq the minimum separation value δ (see Figure 1(b)).

2.2 Machine Delivery Errors

On most current MLCs, the sides of the leaves are designed to have a “tongue-and-groove” (TG) feature (see Figure 2(a)). This design reduces the radiation leakage through the gap between two neighboring MLC leaves [21]. But, it also causes an unwanted underdose and leakage situation when an MLC leaf is used for blocking radiation (see Figures 2(b) and 2(c)). Geometrically, the underdose and leakage error caused by the tongue-and-groove feature associating with an MLC-aperture is a set of 3-D axis-parallel boxes $w \cdot l_i \cdot h$, where w is the (fixed) width of the tongue or groove side of an MLC leaf, l_i is the length of the portion of the i -th leaf that is actually involved in blocking radiation, and $h = \alpha \cdot r$ is the amount of radiation leakage with α being the (fixed) leakage ratio and r being the amount of radiation delivered by that MLC-aperture. Figure 2(b) illustrates the height of the underdose and leakage error, and Figure 2(c) illustrates the width and length of the underdose and leakage error.

We distinguish two types of errors caused by the tongue-and-groove feature of MLC leaves:

Tongue-or-Groove Error: The tongue-or-groove error of an MLC-aperture is defined as the amount of underdose and leakage error occurred whenever the tongue side or groove side of an MLC leaf is used for blocking radiation. The tongue-or-groove error of an IMRT plan (i.e., a set of MLC-apertures) is the sum of the errors of all its MLC-apertures.

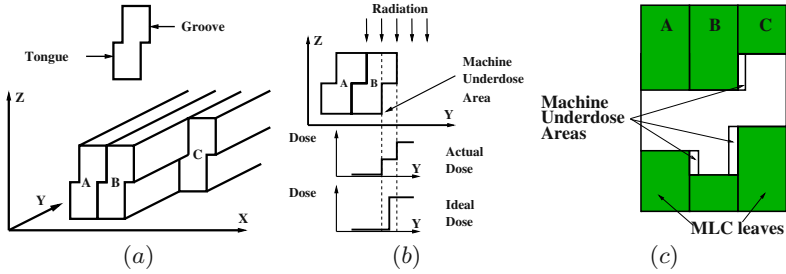


Fig. 2. (a) Illustrating the tongue-and-groove (TG) interlock feature of the MLC in 3-D, where leaf *B* is used for blocking radiation. (b) When leaf *B* is used for blocking radiation, there is an underdose and leakage in the tongue or groove area. (c) The underdose and leakage areas of an MLC-aperture region

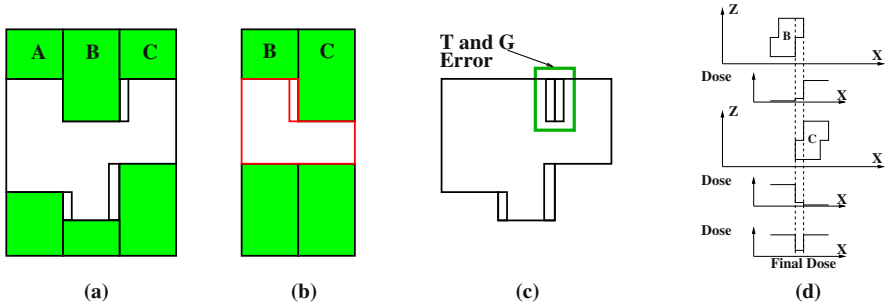


Fig. 3. Illustrating the tongue-and-groove error. (a) and (b): Two MLC-apertures (the shaded rectangles represent MLC leaves). (c) When delivering the two MLC-apertures in (a) and (b) (one by one), the groove side of leaf *B* and tongue side of leaf *C* are both used for blocking radiation, causing a tongue-and-groove error in the area between the leaves *B* and *C*. (d) Illustrating a dose “dip” in the final dose distribution where a tongue-and-groove error occurs

Tongue-and-Groove Error: Unlike the tongue-or-groove error which is defined on each individual MLC-aperture, the tongue-**and**-groove error is defined by the relations between different MLC-apertures. The tongue-and-groove error occurs whenever the tongue side of an MLC leaf and the corresponding groove side of its neighboring leaf are both used for blocking radiation in any two different MLC-apertures of an IMRT plan (see Figure 3). Clearly, the tongue-and-groove error is a subset of the tongue-or-groove error. Note that minimizing the tongue-and-groove error is also important because it usually occurs in the middle of the delivered intensity maps [20], and is actually the most damaging part of the tongue-or-groove error.

Geometrically, for a set of MLC-apertures, the errors caused by using the tongue sides of the MLC leaves for blocking radiation are a set of 3-D axis-parallel boxes, denoted by V_T . Similarly, the errors by using the groove sides of the leaves for blocking radiation are another set of 3-D axis-parallel boxes,

denoted by V_G . Then the tongue-or-groove error for the MLC-aperture set is the sum of the volume values of these two sets (i.e., $|V_T| + |V_G|$), and the tongue-and-groove error is equal to twice the volume value of the intersection between V_T and V_G (i.e., $2 \cdot |V_T \cap V_G|$). We can also view the given IM as a 3-D rectilinear terrain (mountain), denoted by V^* . Then the magnitude of the tongue-or-groove (resp., tongue-and-groove) error can be quantified by the percentage $\frac{|V_T| + |V_G|}{|V^*|}$ (resp., $\frac{|V_T \cap V_G|}{|V^*|}$).

If we view each MLC-aperture as a rectilinear polygonal region on the xy -plane, then the tongue-or-groove error occurs along every vertical boundary edge of this region, except its leftmost and rightmost vertical edges. The leftmost and rightmost vertical edges are excluded (i.e., no error on them) since they are defined by the backup diaphragms along the x -axis, not by the MLC leaves.

2.3 The Static Leaf Sequencing Problem

The 3-D **Static Leaf Sequencing (SLS) Problem** is: Given an IM and an error bound \mathcal{E} , find a set S of MLC-apertures for delivering the IM, such that the machine delivery error (i.e., either the tongue-or-groove error or the tongue-and-groove error) is $\leq \mathcal{E}$ and the size $|S|$ is minimized.

Interchangeably, we also call such MLC-apertures the **B-segments** [7] (for **B**-segments). Each B-segment is of a rectilinear x -monotone polygonal shape of a uniform height $h \geq 1$ (h is the number of dose units delivered by the MLC-aperture).

A key special case of the 3-D SLS problem is the **basic 3-D SLS** problem [4, 5, 6]. This case is similar to the general 3-D SLS problem, except that the height of each of its B-segments must be one. Note that in the general 3-D SLS problem, the uniform height of each B-segment can be any integer ≥ 1 . Studying the basic case is important because the maximum heights of the majority of IMs used in the current clinical treatments are around 5, and an optimal solution for the basic case on such an IM is often very close to an optimal solution for the general case.

3 3-D SLS Algorithms with Error Control

This section presents our SLS algorithms with error control. Due to space limit, we will use our basic 3-D SLS algorithms with error control for the Elekta model to illustrate some of the key ideas of our approach, and leave the details of our complete SLS algorithm suite to the full paper.

3.1 Algorithm for Basic 3-D SLS Problem with Tongue-or-Groove Error Control

Let S be a solution for the basic 3-D SLS problem with tongue-or-groove error control ($S = \{S_i \mid i \in I\}$ is a set of B-segments of height 1 for the given IM, where I is an index set). Consider a B-segment $S_i \in S$. Observe that since each

B-segment S_i has a unit height and an x -monotone rectilinear simple polygonal shape, S_i actually builds a continuous block of a unit height on every IM column C_j that intersects the projection of S_i on the IM grid [7]. We denote such a continuous block by an interval $B_{i,j}$ on the column C_j . (Interchangeably, we also call $B_{i,j}$ a block). Note that when delivering a B-segment S_i , each of its blocks $B_{i,j}$ is delivered by the pair of MLC leaves that is aligned with C_j .

Let $B(S_i) = \{B_{i,j} \mid B_{i,j} = S_i \cap C_j, j = e_i, e_i + 1, \dots, k_i\}$ be the set of blocks that form S_i , where S_i “runs” consecutively from the IM columns C_{e_i} to C_{k_i} . As discussed in Section 2.2, the tongue-or-groove error of S_i occurs along every vertical boundary edge of the polygonal region of S_i , except its leftmost and rightmost vertical edges. Let $|B_{i,j} \oplus B_{i,j+1}|$ denote the length of the symmetric difference between the two intervals of $B_{i,j}$ and $B_{i,j+1}$, $j = e_i, e_i + 1, \dots, k_i - 1$. Thus, the total tongue-or-groove error $TorG(S_i)$ of S_i is the sum of a set of 3-D error volumes, $TorG(S_i) = \sum_{j=e_i}^{k_i-1} w \cdot l_{i,j} \cdot h_i$, where w is the (fixed) width of the tongue or groove side of an MLC leaf, $l_{i,j} = |B_{i,j} \oplus B_{i,j+1}|$ is the length of the leaf portion that is actually used for blocking radiation between blocks $B_{i,j}$ and $B_{i,j+1}$, and $h_i = \alpha \cdot r_i$ is the amount of radiation leakage associated with S_i with r_i being the “height” of S_i . Since in the basic 3-D SLS problem, the B-segments are all of a height one (i.e., $r_i = 1, \forall i \in I$), the tongue-or-groove error of S is $TorG(S) = w \cdot \alpha \cdot \sum_{i \in I} \sum_{j=e_i}^{k_i-1} l_{i,j}$. Observe that $\sum_{j=e_i}^{k_i-1} l_{i,j}$ is the sum of the lengths of all non-extreme vertical edges of the B-segment S_i (e.g., see Figure 2(c)).

Thus, we have the following geometric version of the basic 3-D SLS problem with tongue-or-groove error control: Given an IM and an error bound \mathcal{E}^* , find a set $S = \{S_i \mid i \in I\}$ of B-segments of height one, such that the value $\mathcal{C}(S) = \sum_{i \in I} \sum_{j=e_i}^{k_i-1} l_{i,j} \leq \mathcal{E}^*$ and the size $|S|$ is minimized. Note that here, $\mathcal{E}^* = \mathcal{E}/(w \cdot \alpha)$ for the error bound \mathcal{E} in the definition of the SLS problems in Section 2.3.

For each B-segment $S_i \in S$, now let $B(S_i) = \{B_{i,j} \mid j = 1, 2, \dots, n\}$, such that n is the number of columns of the given IM and some intervals $B_{i,j}$ may be empty. Then we have $\mathcal{C}(S) = \sum_{i \in I} \sum_{j=e_i}^{k_i-1} |B_{i,j} \oplus B_{i,j+1}| = \sum_{j=1}^{n-1} \sum_{i \in I} |B_{i,j} \oplus B_{i,j+1}|$. Note that for each $j = 1, 2, \dots, n-1$, the value $\sum_{i \in I} |B_{i,j} \oplus B_{i,j+1}|$ is actually the tongue-or-groove error for “stitching” the two block-sets $BS(C_j)$ and $BS(C_{j+1})$ for the IM columns C_j and C_{j+1} to form the B-segments as defined by S . Suppose g pairs of blocks are stitched together by S between $BS(C_j)$ and $BS(C_{j+1})$. To minimize the error of S , the error incurred for such a stitching configuration (i.e., $\sum_{i \in I} |B_{i,j} \oplus B_{i,j+1}|$) must be the smallest among all possible stitching configurations with exactly g stitched block pairs between $BS(C_j)$ and $BS(C_{j+1})$ defined by S .

Now we can relate the tongue-or-groove error to the number of B-segments, by associating an error to each stitching configuration between the block-sets for any two consecutive IM columns. Specifically, for any two block-sets $BS(C_j)$ and $BS(C_{j+1})$ for columns C_j and C_{j+1} and each $g = 1, 2, \dots, |M_j|$, where M_j is a maximum size matching between $BS(C_j)$ and $BS(C_{j+1})$, we stitch together exactly g pairs of blocks with the minimum total error. Note that every stitching configuration of exactly g block pairs between $BS(C_j)$ and $BS(C_{j+1})$ with the

minimum error corresponds to a matching of exactly g pairs of intervals with the minimum total weight between the two interval sets of $BS(C_j)$ and $BS(C_{j+1})$. We call such a bipartite matching of intervals (subject to the MLC constraints) an g -matching. Hence, an optimal solution for the basic 3-D SLS problem with error control is specified by a list of block-sets (one for each IM column) and an optimal g_j -matching between two such block-sets $BS(C_j)$ and $BS(C_{j+1})$ for any consecutive columns C_j and C_{j+1} (for some value g_j).

To find the sought block-sets and g_j -matchings, we construct the following directed acyclic graph G^* : (1) Generate all distinct block-sets for each IM column, and let every vertex of G^* correspond to exactly one such block-set. (2) For any two vertices of G^* corresponding to two block-sets $BS(C_j)$ and $BS(C_{j+1})$ for two consecutive IM columns C_j and C_{j+1} , compute a minimum weight g -matching for each $g = 1, 2, \dots, |M_j|$, where M_j is a maximum size matching between $BS(C_j)$ and $BS(C_{j+1})$. For each such g -matching, put a left-to-right edge between the two vertices in G^* , and assign the edge a weight of $(|BS(C_{j+1})| - g)$ and a cost equal to the minimum weight of the g -matching. (3) Add a source vertex s to G^* and connect s to all block-sets for the first IM column; add a sink t and connect all block-sets for the last IM column to t (all edges added here have weight zero and cost zero).

Lemma 1. *Let s, t be the source and sink vertices of G^* . Then, there exists a k -weight shortest s -to- t path in G^* with a total cost $\leq \mathcal{E}^*$.*

Lemma 1 implies that for the basic 3-D SLS problem with a given error bound \mathcal{E}^* , we can obtain a minimum B-segment set for the given IM subject to this error bound, by finding a minimum weight s -to- t path in G^* with a total cost $\leq \mathcal{E}^*$. This is called the *minimum B-segment set problem* [12].

To compute such a set of B-segments, several issues must be resolved: (1) How to generate all distinct block-sets for each IM mountain slice? (2) How to compute an optimal g -matching between two block-sets? (3) How to find a minimum weight s -to- t path in G^* with a total cost $\leq \mathcal{E}^*$?

To generate all distinct block-sets for an IM column, we use the algorithm in [14], whose running time is linear in terms of the number of output block-sets. To find a desired k -weight shortest s -to- t path in G^* , we use Lawler’s dynamic programming algorithm for constrained shortest paths [12]. The sought path can be easily obtained from the dynamic programming table once it becomes available.

Now we show how to compute the optimal g -matchings. Given two block-sets BS_r and BS_b for any two consecutive IM columns, we construct a bipartite graph $G = (R \cup B, E)$ as follows: Each block in BS_r (resp., BS_b) corresponds to a red (resp., blue) vertex in G , and every g -matching between BS_r and BS_b corresponds to an edge between the corresponding red and blue vertices. Here, a red block and a blue block are *compatible* if they satisfy the (machine model specific) MLC constraints. For each edge $e(u, v)$ in G , let their corresponding intervals be $I_u = [l_u, r_u]$ and $I_v = [l_v, r_v]$; then the cost of $e(u, v)$ is assigned as $|l_u - l_v| + |r_u - r_v|$, i.e., the length of the symmetric difference between I_u and I_v .

To compute an optimal g -matching in the bipartite graph G for each $g = 1, 2, \dots, |M|$, where M is the maximum size matching of G , we transform G into a unit-capacity flow network and formulate the task as a minimum cost flow problem with a flow value g . The $|M|$ optimal g -matchings can be computed efficiently by the successive shortest path algorithm [2], i.e., at the end of the g -th stage of the algorithm ($1 \leq g \leq |M|$), a desired optimal g -matching is produced. Since the single source shortest paths in G at each stage can be computed in $O(m + n \log n)$ time, the total time for computing all the $|M|$ optimal g -matchings ($1 \leq g \leq |M|$) is $O(|M|(m + n \log n))$, where $m = |E|$ and $n = |R \cup B|$.

Theorem 1.

$$O\left(\sum_{j=1}^{n-1} \Pi_j \cdot \Pi_{j+1} \cdot \Gamma + \mathcal{K}\right) \cdot n \cdot C_j \cdot \Gamma \cdot \mathcal{K} \cdot g \cdot k \cdot s \cdot t \cdot G^*$$

3.2 3-D Basic SLS Algorithms with Tongue-and-Groove Error Control

As discussed in Section 2.2, the tongue-and-groove error for an IMRT plan is the intersection $|V_T \cap V_G|$, where V_T (resp., V_G) is the error set (i.e., a set of 3-D axis-parallel boxes) caused by the tongue sides (resp., groove sides) of the MLC leaves. At first sight, due to the nonlinearity, it might appear that the tongue-and-groove error is much harder to handle than the tongue-or-groove error. Interestingly, we are able to show that handling the tongue-and-groove error is in fact equivalent to handling the tongue-or-groove error. The key to our solution is the next lemma which implies that our SLS algorithms for tongue-or-groove error control are also applicable to the case of tongue-and-groove error control.

Lemma 2.

$$\mathcal{M} \cdot S \cdot \mathcal{F}(\mathcal{M}) \cdot \mathcal{M}$$

4 Implementation and Experiments

To further examine the clinical feasibility of our new 3-D SLS algorithms, we implemented them using C on the Linux systems and conducted extensive experimental studies using real medical data. We performed Monte Carlo simulations [16] on the treatment plans computed by our software, which proved the correctness of our algorithms/software. We also compared our new 3-D SLS algorithms with the current most popular commercial planning system CORVUS (version 5.0), which showed significant improvements. E.g., on a head and neck cancer

case consisting of 7 IMs, to eliminate tongue-and-groove error, CORVUS would need 262 B-segments, in contrast to the 142 B-segments computed by our new SLS program.

References

1. R.K. Ahuja and H.W. Hamacher. Minimizing Beam-on Time in Radiation Therapy Treatment Planning Using Network Flows. *submitted to Networks*.
2. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Inc., 1993.
3. N. Boland, H.W. Hamacher, and F. Lenzen. Minimizing Beam-on Time in Cancer Radiation Treatment Using Multileaf Collimators. Report, Department of Mathematics, University Kaiserslautern, 2002.
4. T.R. Bortfeld, A.L. Boyer, W. Schlegel, D.L. Kahler, and T.L. Waldron. Realization and Verification of Three-Dimensional Conformal Radiotherapy with Modulated Fields. *Int. J. Radiat. Oncol. Biol. Phys.*, 30:899–908, 1994.
5. T.R. Bortfeld, D.L. Kahler, T.J. Waldron, and A.L. Boyer. X-ray Field Compensation with Multileaf Collimators. *Int. J. Radiat. Oncol. Biol. Phys.*, 28:723–730, 1994.
6. A.L. Boyer. Use of MLC for Intensity Modulation. *Med. Phys.*, 21:1007, 1994.
7. D.Z. Chen, X.S. Hu, S. Luan, C. Wang, and X. Wu. Geometric Algorithms for Static Leaf Sequencing Problems in Radiation Therapy. In *Proc. of 19th ACM Symposium on Computational Geometry (SoCG'03)*, pages 88–97, 2003.
8. D.Z. Chen, X.S. Hu, S. Luan, X. Wu, and C.X. Yu. Optimal Terrain Construction Problems and Applications in Intensity-Modulated Radiation Therapy. In *Lecture Notes in Computer Science, Springer-Verlag, Proc. 10th Annual European Symposium on Algorithms (ESA'02)*, volume 2461, pages 270–283, 2002.
9. J. Dai and Y. Zhu. Minimizing the Number of Segments in a Delivery Sequence for Intensity-Modulated Radiation Therapy with Multileaf Collimator. *Med. Phys.*, 28(10):2113–2120, 2001.
10. J. Deng, T. Pawlicki, Y. Chen, J. Li, S.B. Jiang, and C.-M. Ma. The MLC Tongue-and-Groove Effect on IMRT Dose Distribution. *Physics in Medicine and Biology*, 46:1039–1060, 2001.
11. T.J. Jordan and P.C. Williams. The Design and Performance Characteristics of a Multileaf Collimator. *Phys. Med. Biol.*, 39:231–251, 1994.
12. E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.
13. F. Lenzen. An Integer Programming Approach to the Multileaf Collimator Problem. Master's thesis, University of Kaiserslautern, June 2000.
14. S. Luan, D.Z. Chen, L. Zhang, X. Wu, and C.X. Yu. An Optimal Algorithm for Computing Configuration Options of One-dimensional Intensity Modulated Beams. *Phys. Med. Biol.*, 48(15):2321–2338, 2003.
15. S. Luan, C. Wang, S.A. Naqvi, D.Z. Chen, X.S. Hu, C.L. Lee, and C.X. Yu. A New MLC Segmentation Algorithm/Software for Step and Shoot IMRT. *Med. Phys.*, 31(4):695–707, 2004.
16. S.A. Naqvi, M. Earl, and D. Shepard. Convolution/Superposition Using the Monte Carlo Method. *Phys. Med. Biol.*, 48(14):2101–2121, 2003.
17. R.A.C. Siochi. Minimizing Static Intensity Modulation Delivery Time Using an Intensity Solid Paradigm. *Int J. Radiat. Oncol. Biol. Phys.*, 43(3):671–680, 1999.

18. S. Webb. *The Physics of Three-Dimensional Radiation Therapy*. Bristol, Institute of Physics Publishing, 1993.
19. S. Webb. *The Physics of Conformal Radiotherapy — Advances in Technology*. Bristol, Institute of Physics Publishing, 1997.
20. P. Xia and L.J. Verhey. MLC Leaf Sequencing Algorithm for Intensity Modulated Beams with Multiple Static Segments. *Med. Phys.*, 25:1424–1434, 1998.
21. C.X. Yu. Design Considerations of the Sides of the Multileaf Collimator. *Phys. Med. Biol.*, 43(5):1335–1342, 1998.

An Efficient Exact Algorithm for the Minimum Ultrametric Tree Problem

Hsin-Fu Chen and Maw-Shang Chang

Department of Computer Science and Information Engineering,
National Chung Cheng University,
Ming-Shiun, Chiayi 621, Taiwan
{chf91, mschang}@cs.ccu.edu.tw

Abstract. The minimum ultrametric tree problem is an NP-hard evolutionary tree construction problem. Wu et al. proposed a branch-and-bound algorithm that solves the minimum ultrametric tree problem to optimality in [11]. In this paper, we use a look-ahead approach to computing a tighter lower bound for a subproblem. Besides we propose to use the persistent data structure to speedup the branch-and-bound algorithm. Experimental results show that our algorithm outperforms the previous one. We believe that the approach used in the paper can be adapted to speedup other branch-and-bound algorithms.

1 Introduction

Evolutionary trees (also called phylogenetic trees, or more simply, phylogenies) are useful tools to model evolutionary history (or relationship) of a set of species. According to the following two common assumptions of evolution: (1) All present-day species were evolved from one common ancestor, and (2) The rate of evolution is constant, we can model the evolutionary relationship of a set of present-day species by an edge-weighted binary tree that satisfies the following properties: (1) Leaves represent present-day species. (2) Internal nodes represent hypothesis ancestors of present-day species, and the root represents the common ancestor from which all species evolved. (3) Weights of edges represent the evolutionary distances, e.g., evolutionary time, between the corresponding two species. Thus, all weights must be greater than or equal to zero. (4) For any subtree, the lengths of the paths from the root to its leaves are all the same, where the length of a path is defined to be the sum of weights of edges of the path.

Given a set of species and the evolutionary distance between every two species, constructing an “optimal” evolutionary tree for those species is an important problem in computational biology. The constructed evolutionary tree must satisfy the basic criterion: For any two species, the length of the path connecting the two species in the evolutionary tree must be greater than or equal to the given evolutionary distance between them. In this paper, we deal with a special type of evolutionary trees satisfying an extra criterion: The sum of weights of all edges is minimized. We call a tree satisfying this criterion a *minimum ultrametric tree*.

or \mathcal{S}_n for short. See [9] for an introduction to evolutionary trees in general, and [4, 5] for several other models of evolutionary trees.

Given a set of species and the evolutionary distance between every two species, the Δ MUT problem, or the Δ for short, is to find a minimum ultrametric tree for those species. The MUT problem will be formally defined in Section 2. Farach et al. proved that the MUT problem is NP-hard, and there exists an ϵ such that the MUT problem cannot be approximated in polynomial time within ratio n^ϵ unless $P = NP$, where n is the number of species [4].

There is a variation of the MUT problem—the Δ MUT problem, or the Δ for short. The problem is introduced by Wu et al. in [11]. The Δ MUT problem has the same definition as the MUT problem, except the input is a metric, i.e., the distances satisfy the triangle inequality. Wu et al. proved that the Δ MUT problem is NP-hard and can be approximated in polynomial time with error ratio $1.5(1 + \lceil \log n \rceil)$ [11]. Later, Huang et al. improved the result by proposing another polynomial time approximation algorithm, with error ratio $\leq \log_\alpha n + 1 \cong 1.44 \lceil \log n \rceil + 1$, where $\alpha = (\sqrt{5} + 1)/2$ [6].

Wu et al. proposed a branch-and-bound algorithm that solves the MUT problem, as well as the Δ MUT problem [11]. Their algorithm can solve the MUT problem instances and Δ MUT problem instances of size no greater than 19 and 25 species, respectively, in reasonable time. In this paper, we improve the branch-and-bound algorithm by giving a new algorithm for computing a tighter lower bound on the optimal value of a subproblem and adopting a persistent data structure [3] to store subproblems. Using a persistent data structure for storing subproblems allows us to compute a tighter lower bound on the optimal value of a subproblem and to generate subproblems efficiently.

The rest of this paper is organized as follows: In Section 2, we formally define the MUT problem and review the branch-and-bound algorithm for the MUT problem proposed by Wu et al. [11]. In Section 3, we propose an algorithm for computing a tighter lower bound on the optimal value of a subproblem and introduce a persistent data structure for storing subproblems. Finally, in Section 4, we show the experimental results of our implementation, compared with the implementation by Wu et al. [11].

2 Preliminaries

First we formally define the MUT problem.

Definition 1. Let $\mathcal{S}_n = \{1, 2, \dots, n\}$ be a set of n species, $n \geq 2$. Let M be an $n \times n$ matrix such that $M[i, j] = M[j, i]$ and $M[i, j] \geq 0$ for $1 \leq i, j \leq n$. Let $M[i, i] = 0$ for $1 \leq i \leq n$.

In the rest of the paper, $\mathcal{S}_n = \{1, 2, \dots, n\}$ is a set of n species, S is a subset of \mathcal{S}_n , and M is a distance matrix of \mathcal{S}_n . Besides, all trees mentioned are binary trees if we do not explicitly specify.

Definition 2. Let $T = (V(T), E(T))$ be a tree with root r . For any node $v \in V(T)$, let T_v be the subtree rooted at v . Let $L(T)$ be the set of leaves of T .

Definition 3. Let $T = (V, E)$ be a tree with root r and weight function w . For any node $s \in V$, let T_s be the subtree rooted at s . Let $w(T, s) = \sum_{e \in E(T_s)} w(e)$ be the weight of the subtree T_s . Let $w(T) = \sum_{e \in E} w(e)$ be the total weight of T . Let $w(T, r)$ be the weight of the subtree rooted at r .

Definition 4. Let $T = (V, E)$ be a tree with root r and weight function w . Let $u, v \in V$ be two nodes. Let $d_T(u, v) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$ be the distance between u and v in T , where $\langle v_0 = u, v_1, \dots, v_k = v \rangle$ is a path from u to v . Let $d_T(v, v) = 0$.

Definition 5. Let $T = (T, w)$ be a tree with root r and weight function w . Let $d_T(r, i) = d_T(r, j)$ for $i, j \in L(T)$.

Let $\mathcal{T} = (T, w)$ be an ultrametric tree. It is easy to see that, for any internal node v , (T_v, w') , where $w'(e) = w(e)$ for $e \in E(T_v)$, is an ultrametric tree too.

Definition 6. Let S and M be two metrics on $L(T)$. Let $d_{\mathcal{T}}(i, j) \geq M[i, j]$ for $i, j \in L(T)$. Let $L(T) = S$ be the topology of \mathcal{T} . Let S and M be two metrics on $L(T)$.

Definition 7. Let S be a metric on $L(T)$. Let S_n be the minimum ultrametric tree problem. Let MUT problem be the minimum ultrametric tree problem.

In the rest of this section, we review some properties about (minimum) ultrametric trees and the branch-and-bound algorithm in [11]. The algorithm is listed in Algorithm 1 for reference. See [11] for details of the algorithm, such as the maxmin permutation, UPGMM, and the modified best-first search strategy, which we will not explain here because of the limitation of space.

Theorem 1. [11] $M[1, 2] \geq M[i, j]$ for $1 \leq i, j \leq n$. Let $\mathcal{T} = (T, w)$ be a tree with root r and weight function w . Let $1 \in L(T_u)$ and $2 \in L(T_v)$. Let u and v be two nodes in T .

Definition 8. Let $T = (V, E)$ be a tree with root r and weight function w . Let $L(T) = S$ be the topology of T . Let M be a metric on $L(T)$. Let $MUT(T)$ be the minimum ultrametric tree problem. Let S be a metric on $L(T)$. Let $W(T, s) = w(T, s)$ for $s \in V$. Let $W(T) = w(T)$ be the total weight of T .

Algorithm 1. The branch-and-bound algorithm for the MUT problem

Input: A distance matrix M of \mathcal{S}_n .

Output: A minimum ultrametric tree for M .

- 1: Relabel the species such that $(1, 2, \dots, n)$ is a maxmin permutation.
- 2: Let \mathcal{U} be an ultrametric tree for M obtained by using UPGMM.
- 3: $UB \leftarrow w(\mathcal{U})$.
- 4: $\mathcal{TS} \leftarrow \{T\}$, where T is the leaf-labelled tree with $L(T) = \{1, 2\}$. \triangleright See Theorem 1.
- 5: **while** $\mathcal{TS} \neq \phi$ **do**
- 6: $\mathcal{TS} \leftarrow \{T \in \mathcal{TS} : LB_0(T) < UB\}$.
- 7: Extract a leaf-labelled tree T from \mathcal{TS} by the modified best-first search strategy.
- 8: Assume that $L(T) = \{1, 2, \dots, i\}$.
- 9: Compute $W(T)$ and $LB_0(T)$. \triangleright See Theorem 2 and Theorem 6.
- 10: **if** $i = n$ **then**
- 11: **if** $W(T) < UB$ **then**
- 12: $\mathcal{U} \leftarrow MUT(T)$.
- 13: $UB \leftarrow W(T)$.
- 14: **end if**
- 15: **else if** $LB_0(T) < UB$ **then**
- 16: $\mathcal{TS} \leftarrow \mathcal{TS} \cup I(T, i + 1)$.
- 17: **end if**
- 18: **end while**
- 19: Return \mathcal{U} .

Definition 9. $T = (V, E)$ $L(T) = S, i \in \mathcal{S}_n \setminus S$
 $e = (u, v) \in E, u \dots v, T \dots T' = I(T, e, i)$
 $L(T') = S \cup \{i\}$
 $i \dots T \dots e \dots i \dots e \dots$
 $(u, x) \dots (x, v), \dots (x, i), \dots i \dots i \dots$
 $I(T, i) \dots \{I(T, e, i) : e \in E\}$

Definition 10. $T \dots T'$
 $T \subset T', T' = I(T, e, i) \dots e \dots T \dots i \in \mathcal{S}_n \setminus L(T) \dots T'$
 T
 T' contains T

Definition 11. $T \dots L(T) = S$
 $\mathcal{S}_n \dots T \dots \mathcal{S}_n \dots T$
 $\mathcal{S}_n \dots T' \dots T \dots w(T) \dots T = (T', w)$
 $\mathcal{S}_n \dots T$

Definition 12. [11] lower bound array M, \dots, LBA, \dots
 $LBA[n] = 0, \dots, i = 2, 3, \dots, n - 1, LBA[i] \dots$
 $\sum_{j=i+1}^n \min\{M[j, k]/2 : k \in \{1, 2, \dots, j - 1\}\}$

Theorem 2. [11] $T \dots L(T) = \{1, 2, \dots, i\}, \dots$
 $2 \leq i < n \dots T \dots \mathcal{S}_n \dots T, \dots w(T) \geq$
 $LB_0(T) = W(T) + LBA[i]$

The above theorem provides a lower bound for the branch-and-bound algorithm given in [11]. Clearly the lower bound array can be computed in $O(n^2)$ time in the preprocessing stage. We now review the $O(n^2)$ -time algorithm used in [11] for computing $W(T)$. In other words, the running time of the algorithm used in [11] for computing $LB_0(T)$ is $O(n^2)$.

Definition 13. Let $\mathcal{T} = (T, w)$ be an ultrametric tree with $T = (V, E)$ and root r . For every vertex $v \in V$, let $h(\mathcal{T}, v)$ be the height of v in \mathcal{T} . For every vertex $v \in V$, let T_v be the subtree of \mathcal{T} rooted at v . For every vertex $u \in L(T_v)$, let $h(\mathcal{T}, v, u) = d_{\mathcal{T}}(v, u)$. For every vertex $v \in L(T)$, let $h(\mathcal{T}, v) = 0$.

The following two theorems show the properties of the heights of vertices in an ultrametric tree and the relationship between the weight of an ultrametric tree and the heights of all vertices in the ultrametric tree.

Theorem 3. [11] Let $\mathcal{T} = (T, w)$ be an ultrametric tree with $T = (V, E)$ and root r . Then $w(T) = h(\mathcal{T}, r) + \sum_{v \in V} h(\mathcal{T}, v)$.

Let $\mathcal{T} = (T, w)$ be an ultrametric tree with $T = (V, E)$ and root r . Immediately follows from the above theorem, we have that $w(T) = 2h(\mathcal{T}, r) + \sum_{v \in V \setminus \{r\}} h(\mathcal{T}, v)$ and $\sum_{v \in V} h(\mathcal{T}, v) = w(T) - h(\mathcal{T}, r)$.

Definition 14. Let $T = (V, E)$ be a tree with $L(T) \subseteq \mathcal{S}_n$. Let $\mathcal{T} = MUT(T)$ be the minimum ultrametric tree of T . For every vertex $v \in V$, let $H(\mathcal{T}, v) = h(\mathcal{T}, v)$.

Theorem 4. [11] Let $T = (V, E)$ be a tree with $L(T) \subseteq \mathcal{S}_n$. Let $\mathcal{T} = MUT(T)$. For every vertex $v \in V \setminus L(T)$, $H(\mathcal{T}, v) = \frac{1}{2} \max\{M[i, j] : i, j \in L(T_v)\}$.

Definition 15. Let $A, B \subseteq \mathcal{S}_n$ be two subsets of \mathcal{S}_n with $A \cap B = \phi$. Let $md(A, B)$ be the maximum distance between a vertex in A and a vertex in B . Let $md(A, \{j\}) = \frac{1}{2} \max\{M[i, j] : i \in A, j \in B\}$ and $md(A, j) = \frac{1}{2} \max\{M[i, j] : i \in A, j \in \mathcal{S}_n \setminus A\}$.

Immediately following from Theorem 4, we have the following theorem

Theorem 5. Let T be a tree with $L(T) \subseteq \mathcal{S}_n$ and $\mathcal{T} = MUT(T)$. For every vertex $s \in T$, let $H(\mathcal{T}, s) = \max\{H(\mathcal{T}, u), H(\mathcal{T}, v), md(L(T_u), L(T_v))\}$.

Given a leaf-labelled tree T with $L(T) \subseteq \mathcal{S}_n$, we compute a minimum ultrametric function for T with respect to M as follows. First we compute $H(\mathcal{T}, v)$ for every internal node v in T from M by a postorder traversal of tree T according to Theorem 5. This can be done in $O(n^2)$ time. For all $e = (u, v) \in E(T)$, where u is the parent of v , let $w(e)$ be $H(\mathcal{T}, u) - H(\mathcal{T}, v)$. Therefore, we have the following theorem.

Theorem 6. [11] Let $T = (V, E)$ be a tree with $L(T) \subseteq \mathcal{S}_n$. Let $\mathcal{T} = MUT(T)$. For every vertex $v \in V$, let $w(T, v) = H(\mathcal{T}, v)$. Then $W(T) = \sum_{v \in V} w(T, v)$ and $W(T)$ can be computed in $O(n^2)$ time.

Given the minimum ultrametric function of T with respect to M , $W(T)$ can be computed in $O(n)$ time. Given T , $W(T)$ can be computed in $O(n^2)$ time.

3 The Efficient Implementation

In Algorithm 1, $W(T)$ and $LB_0(T)$ are computed for each tree T generated in $O(n^2)$ time. To improve the branch-and-bound algorithm, we propose a tighter lower bound $LB_1(T)$ and show how to compute it in $O(n)$ time. A revised version of Algorithm 1 is sketched in Algorithm 2.

Algorithm 2. The revised branch-and-bound algorithm for the MUT problem

Input: A distance matrix M of S_n .

Output: A minimum ultrametric tree for M .

```

1: Relabel the species such that  $(1, 2, \dots, n)$  is a maxmin permutation.
2: Let  $\mathcal{U}$  be an ultrametric tree for  $M$  obtained by using UPGMM.
3:  $UB \leftarrow w(\mathcal{U})$ .
4:  $\mathcal{TS} \leftarrow \{T\}$ , where  $T$  is the leaf-labelled tree with  $L(T) = \{1, 2\}$ .  $\triangleright$  See Theorem 1.
5: while  $\mathcal{TS} \neq \phi$  do
6:   Extract a tree  $T$ , from  $\mathcal{TS}$  by the depth-first search strategy.
7:   Assume that  $L(T) = \{1, 2, \dots, i\}$ .
8:   if  $i = n$  then
9:     if  $W(T) < UB$  then
10:       $\mathcal{U} \leftarrow T$ .
11:       $UB \leftarrow W(T)$ .
12:     end if
13:   else
14:     Compute  $md(L(T_v), i + 1)$  for every node  $v$  of  $T$ .
15:     for every edge  $e \in E$  do
16:        $T' \leftarrow I(T, e, i + 1)$ .
17:       Let  $x$  be the vertex in  $T'$  generated by the insertion that divides edge  $e$ .
18:       Compute  $H(T', v)$  for every node  $v \in \{x\} \cup \{v : v \text{ is an ancestor of } x\}$ .
19:       Compute  $\hat{H}(T', v)$  for every internal node  $v$  of  $T'$ .
20:       if  $i = n - 1$  then
21:          $LB \leftarrow W(T')$ .
22:       else
23:         Compute  $md(L(T'_v), i + 2)$  for every node  $v$  of  $T'$ .
24:         Compute  $W(T'_v, i + 2)$  for every node  $v$  of  $T'$ .
25:          $LB \leftarrow W(T', i + 2) + LBA[i + 2]$ .  $\triangleright$  See Theorem 7.
26:       end if
27:       if  $LB < UB$  then
28:          $\mathcal{TS} \leftarrow \mathcal{TS} \cup \{T'\}$ .
29:       end if
30:     end for
31:   end if
32: end while
33: Return  $\mathcal{U}$ .

```

3.1 The New Lower Bound

We use a look-ahead approach to obtaining a better lower bound. We show this approach can be implemented efficiently. We believe that this approach can be

adapted to obtain tighter lower bounds for other branch-and-bound algorithms. We first describe the new lower bound $LB_1(T)$ and then show how to compute $LB_1(T)$ in $O(n)$ time.

Definition 16. Let $T = (V, E)$ be a tree with root r and $L(T) \subset S_n$, $i \in S_n \setminus L(T)$, $v \in V(T)$. Then

$$MW(T, v, i) = \min\{W(I(T, e, i)) : e \in E(T_v)\} + MW(T, i)$$

and

$$MW(T, r, i) = MW(T, v, i) = \infty \quad v \in L(T)$$

By the above definition, the following lemma can be verified easily.

Lemma 1. Let $T = (V, E)$ be a tree with root r and $L(T) \subset S_n$, $i \in S_n \setminus L(T)$, $r, u, v \in V(T)$. Then

$$MW(T, i) = \min\{W(I(T, (r, u), i)), W(I(T, (r, v), i)), MW(T, u, i), MW(T, v, i)\}$$

Theorem 7. Let $T = (V, E)$ be a tree with root r and $L(T) = \{1, 2, \dots, i - 1\}$, $2 \leq i < n$, $T \in \mathcal{S}_n$. Then

$$w(T) \geq LB_1(T) = MW(T, i) + LBA[i]$$

By definition, there exists an edge $e \in E$ such that T contains $I(T, e, i)$ and $W(I(T, e, i)) \geq MW(T, i)$. By Theorem 2, we have $w(T) \geq W(I(T, e, i)) + LBA[i] \geq MW(T, i) + LBA[i]$. \square

Lemma 2. Let $T = (V, E)$ be a tree with root r and $L(T) \subset S_n$, $i \in S_n \setminus L(T)$, $(u, v) \in E$, $v \in V(T)$, $u, x \in S_n$. Then

$$H(T', s) = \begin{cases} \max\{H(T, s), md(L(T_s), i)\} & s \neq x, \\ \max\{H(T, v), md(L(T_v), i)\} & s = x, \\ H(T, s) & \end{cases}$$

where $T' = (V, E) \setminus \{(u, v)\}$ and $L(T') = L(T) \cup \{i\}$. \square

We first prove the case that s is an ancestor of x holds. By Theorem 4, we have

$$\begin{aligned} H(T', s) &= \max\{M[k, l]/2 : k, l \in L(T')\} \\ &= \max\{M[k, l]/2 : k, l \in L(T_s) \cup \{i\}\} \\ &= \max\{M[k, l]/2, M[m, i]/2 : k, l, m \in L(T_s)\} \\ &= \max\{H(T, r), md(L(T_s), i)\}. \end{aligned}$$

The other two cases can be proved in the same way. \square

Definition 17. Let $T = (V, E)$ be a tree with root r and $L(T) \subseteq S_n$, $v \in V(T)$. Then

$$\tilde{H}(T, v) = W(T, v) - \sum_{s \in V(T_v)} H(T, s)$$

and

$$\tilde{H}(T, v) = W(T, v) - H(T, v)$$

Lemma 3. Let $T = (V, E)$ be a tree with root r and $L(T) \subset S_n$, $i \in S_n \setminus L(T)$, $r, u, v \in V(T)$. Then

$$\begin{aligned} W(I(T, (r, u), i)) &= 2 \max\{H(T, r), md(L(T), i)\} + \max\{H(T, u), md(L(T_u), i)\} \\ &\quad + \tilde{H}(T, u) + \tilde{H}(T, v) \end{aligned}$$

For simplicity use T' for $I(T, (r, u), i)$. Assume that $x \in V(T')$ is the new vertex generated by the insertion that divides edge e . By Lemma 2, we have $H(T', r) = \max\{H(T, r), md(L(T), i + 1)\}$ and $H(T', x) = \max\{H(T, u), md(L(T_u), i)\}$. Therefore, by Theorem 3, the lemma holds. \square

Lemma 4. Let $T = (V, E)$ be a tree with root r , $L(T) \subset S_n$, $i \in S_n \setminus L(T)$, $r = (u, v)$ and $r = (u, \dots)$.

$$MW(T, u, i) = 2 \max\{H(T, r), md(L(T), i)\} + \tilde{H}(T, v) + MW(T_u, i) - \max\{H(T, u), md(L(T_u), i)\}$$

Let $T' \in \mathcal{TS} = \{I(T, e, i) : e \in E(T_u)\}$. By Lemma 2, we know that $H(T', r)$ and $H(T', u)$ are $\max\{H(T, r), md(L(T), i)\}$ and $\max\{H(T, u), md(L(T_u), i)\}$, respectively. By Theorem 3, we have

$$\begin{aligned} W(T') &= 2H(T', r) + \tilde{H}(T', v) + \tilde{H}(T', u) \\ &= 2 \max\{H(T, r), md(L(T), i)\} + \tilde{H}(T, v) + W(T', u) - H(T', u) \\ &= 2 \max\{H(T, r), md(L(T), i)\} + \tilde{H}(T, v) \\ &\quad + W(T', u) - \max\{H(T, u), md(L(T_u), i)\} \end{aligned}$$

By Theorem 4, $H(T', s) = H(T'_u, s)$ for $s \in V(T_u)$. Thus $MW(T_u, i) = \min\{W(T', u) : T' \in \mathcal{TS}\}$ and hence the lemma holds. \square

Theorem 8. Let $T = (V, E)$ be a tree with root r , $L(T) = \{1, 2, \dots, i - 1\}$, $2 \leq i < n$, $H(T, v) = \dots$, $v \in V$, \dots , $md(L(T_s), i)$, $\tilde{H}(T, s)$, $W(T_s, i) = \dots$, $s \in T$. $O(n)$.

First it is easy to see that we can compute $md(L(T_s), i)$ and $\tilde{H}(T, s)$ for every internal node s of T in $O(n)$ time by a postorder traversal of tree T . By Lemma 1, 3, and 4, we can compute $MW(T_s, i)$ for every internal node s of T in $O(n)$ time by a postorder traversal of tree T too. \square

3.2 An Efficient Persistent Data Structure

In Algorithm 2, all trees are stored in \mathcal{TS} . For every internal node v of a tree T , we associate a variable storing $H(T, v)$. To speedup the computation of $I(T, e, i)$ from T and save space, we use a persistent data structure. See [3] for studies on persistent data structures. The persistent data structure used here can let all trees in \mathcal{TS} share the common part. We believe that this data structure is very suitable for the branch-and-bound algorithm to store subproblems generated. By using this data structure, we can save time in generating subproblems and computing their lower bounds.

Figure 1 illustrates how we create $I(T, e, i)$ from T . Assume that $e = (u, v) \in E(T)$, $i \notin L(T)$, $\langle v_0 = v, v_1 = u, v_2, \dots, v_k = r \rangle$ is the path from v to r in T , and

u_j is the sibling of v_j in T for $j = 0, 1, \dots, k - 1$. To create $T' = I(T, e, i)$, we first create $k + 1$ new nodes v'_0, v'_1, \dots, v'_k . Then we let v'_k be the root of T' , v'_{i-1} and u_{i-1} be the two children of v'_i for $1 \leq i \leq k$, and v_0 and i be the two children of v'_0 . Then we compute $H(T', s)$ for $s \in \{v'_0, v'_1, \dots, v'_k\}$ according to Lemma 2. We can see that T' and T share a common part and T' can be obtained from T in $O(n)$ time. The advantage of such implementation is that the number of nodes required for \mathcal{TS} is dramatically reduced. Only at most $\lceil |V(T)|/2 \rceil + 1$ new nodes are created for T' .

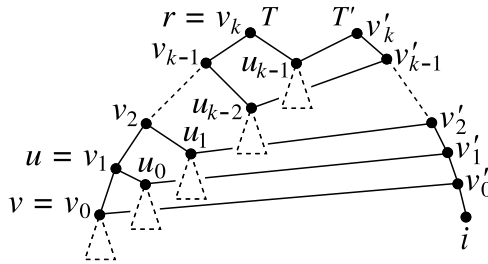


Fig. 1. Creating $T' = I(T, e, i)$ from T

After a tree being extracted according to the search strategy, it should be destroyed. All nodes in the tree cannot be destroyed directly, because some nodes may also be used by other trees. Since there exists no cycle in our data structure, this problem can easily be tackled by applying a well-known technique in garbage collection—*reference counting*. The principle of the technique is quite simple: Each node contains a variable that keeps the number of nodes pointing to it. (We say that node u points to node v if v is a child of u in a tree.) A node can be destroyed only if there exists no node pointing to it.

4 Experimental Results

We implemented Algorithm 2 in C programming language. We compare the performance of our implementation (BBMUT) with the program (BBU) implemented by Wu in [11]. The platform on which the two programs ran is a PC with AMD Athlon 850 MHz CPU. Both BBMUT and BBU were compiled from the source codes using GCC 3.2 with optimization option `-O2` enabled and ran on Red Hat Linux 9. The test data were generated randomly and the distances in the test data are integers between 2 and 100. The running time for non-metric and metric are shown in Fig. 2 and Fig. 3, respectively. Note that y-axes are log-scaled. Experimental results show that our algorithm outperforms the previous one. Larger problem instances can be solved in reasonable time.

To compare the performance of the two lower bounds LB_0 and LB_1 , we implement two branch-and-bound programs. Both of them use the depth-first

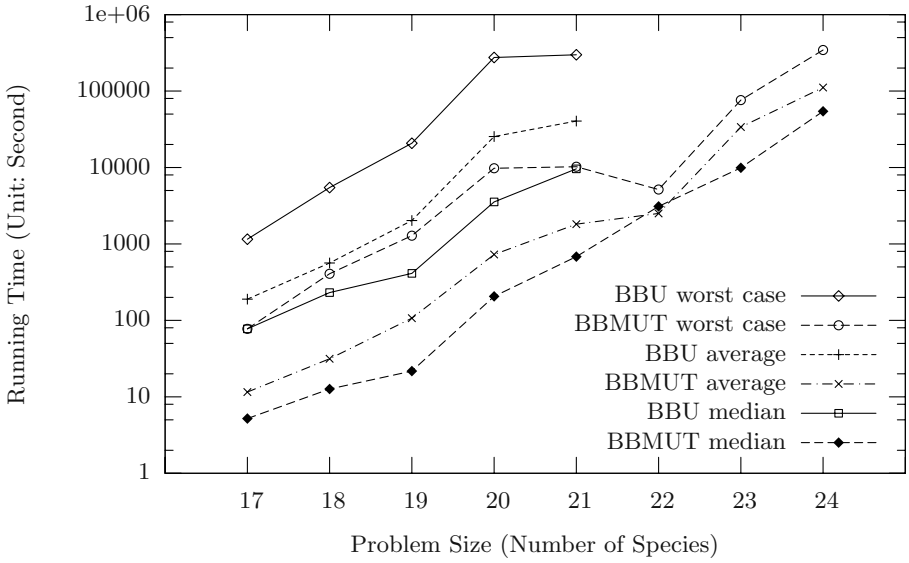


Fig. 2. Running time for non-metric input. (The number of instances for problem size 17 to 24 are 50, 50, 30, 30, 10, 10, 10, and 10, respectively)

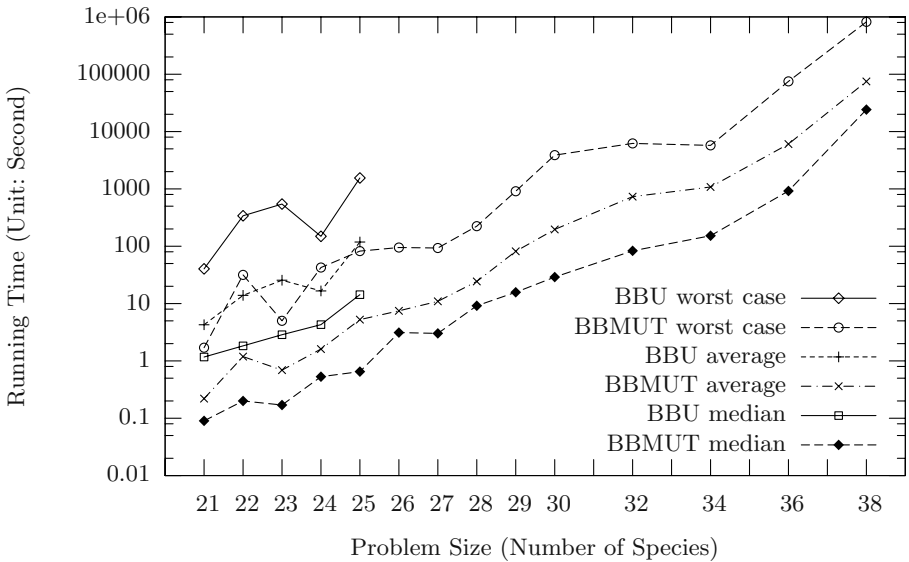


Fig. 3. Running time for metric input. (The number of instances for each problem size is 50)

Table 1. Comparisons of Lower Bounds for non-metric input

The number of trees in millions for non-metric input						
	# of species	17	18	19	20	21
	# of instances	50	50	30	30	10
worst case	LB_0	279.35	1,534.51	3,117.90	4,195.21	3,983.80
	LB_1	120.64	622.91	1,951.28	3,038.54	2,763.47
average	LB_0	46.70	127.46	252.52	1,024.77	2,271.09
	LB_1	18.27	49.01	164.95	541.53	1,479.75
median	LB_0	22.26	49.30	82.49	782.77	2,326.76
	LB_1	8.27	19.85	34.13	321.95	1,056.36

Table 2. Comparisons of Lower Bounds for metric input

The number of trees in millions for metric input										
	# of species	21	22	23	24	25	26	27	28	29
	# of instances	50	50	50	50	50	50	50	50	50
worst case	LB_0	5.05	127.96	13.28	62.58	256.07	274.51	643.98	555.47	2,996.82
	LB_1	2.67	46.66	6.87	21.88	125.23	144.10	138.37	336.39	1,347.37
average	LB_0	0.62	4.23	2.06	2.86	18.45	22.73	43.39	75.25	263.72
	LB_1	0.32	1.77	1.04	2.40	8.03	11.42	16.68	36.56	121.46
median	LB_0	0.22	0.52	0.45	1.40	2.11	9.89	9.53	25.30	47.68
	LB_1	0.12	0.27	0.24	0.78	0.99	4.83	4.65	13.68	22.46

search strategy. But one use LB_0 and the other use LB_1 for bounding. We then count the numbers of trees generated by the two programs, respectively. Table 1 and 2 show the numbers of trees generated by each of them for nonmetric and metric input respectively. From the table we can see that, in most of cases, the number of trees generated by the program using LB_1 for bounding is about half of the number of trees generated by the program using LB_0 for bounding.

Though the program using LB_1 for bounding generates less trees than the one using LB_0 , the running time of the program is still proportional to the running time for generating a tree and computing the LB_1 of the tree. Therefore the $O(n)$ time algorithm for generating a tree and computing its lower bound LB_1 remains the key of the success of program BBMUT.

References

1. S.R. Arikati and C.P. Rangan, Linear algorithm for optimal path cover problem on interval graphs, Inform. Process. Lett. 35 (1990) 149–153.
2. Charles H. Bennett and Ming Li and Bin Ma: Chain letters and evolutionary histories. Scientific American (2003) 76–81
3. James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan: Making data structures persistent. Journal of Computer and System Sciences 38 (1989) 86–124

4. Martin Farach, Sampath Kannan, and Tandy Warnow: A robust model for finding optimal evolutionary trees. *Algorithmica* **13** (1995) 155–179
5. M. D. Hendy and David Penny: Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences* **59** (1982) 277–290
6. Chia-Mao Huang and Chang-Biau Yang: Approximation algorithms for constructing evolutionary trees. In: *Proceedings of 2001 National Computer Symposium, Taipei, Taiwan* (2001) A099–A109
7. C. D. Michener and R. R. Sokal: A quantitative approach to a problem in classification. *Evolution* **11** (1957) 130–162
8. D. Penny and M. D. Hendy and M. A. Steel: Progress with methods for constructing evolutionary trees. *Trends in Ecology and Evolution* **7** (1992) 73–79
9. João Setubal and João Meidanis: *Introduction to computational molecular biology*. PWS Publishing Company (1997)
10. N. Saitou and M. Nei: The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* **4** (1987) 406–425
11. Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang: Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. *Journal of Combinatorial Optimization* **3** (1999) 199–211

On the Range Maximum-Sum Segment Query Problem

Kuan-Yu Chen¹ and Kun-Mao Chao^{1,2}

Department of Computer Science and Information Engineering,
Institute of Networking and Multimedia,
National Taiwan University, Taipei, Taiwan 106
{r92047, kmchao}@csie.ntu.edu.tw

Abstract. We are given a sequence A of n real numbers which is to be preprocessed. In the Range Maximum-Sum Segment Query (RMSQ) problem, a query is comprised of two intervals $[i, j]$ and $[k, l]$ and our goal is to return the maximum-sum segment of A whose starting index lies in $[i, j]$ and ending index lies in $[k, l]$. We propose the first known algorithm for this problem in $O(n)$ preprocessing time and $O(1)$ time per query under the unit-cost RAM model. We also use the RMSQ techniques to solve three relevant problems in linear time. These variations on the basic theme demonstrate the utilities of the techniques developed in this paper.

Keywords: Algorithm, RMQ, maximum-sum interval, sequence analysis.

1 Introduction

Sequence analysis in bioinformatics has been studied for decades. One important line of investigation in sequence analysis is to locate the biologically meaningful segments, like conserved regions or GC-rich regions in DNA sequences. A common approach is to assign a real number (also called scores) to each residue, and then look for the maximum-sum or maximum-average segment [3, 4, 9, 10].

Huang [8] used the expression $x - p \cdot l$ to measure the GC richness of a region, where x is the C+G count of the region, p is a positive constant ratio, and l is the length of the region. Huang [8] extended the well-known recurrence relation used by Bentley [2] for solving the maximum-sum consecutive subsequence problem, and derived a linear time algorithm for computing the optimal segments of lengths at least L . Lin, Jiang, and Chao [10] and Fan *et al.* [4] studied the maximum-sum segment problem of length at least L and at most U . Ruzzo and Tompa [11] gave a linear time algorithm for finding all maximal-sum subsequences. Wang and Xu [14] proposed a linear time algorithm for finding the longest segment with lower-bound average.

In this paper, we consider a more general problem in which we wish to find the maximum-sum segment whose starting and ending indices lie in given intervals. By preprocessing the sequence in $O(n)$ time, each query can be answered in $O(1)$ time. We also show that it can be easily utilized to yield alternative

linear-time algorithms for finding the maximum-sum segment with length constraints, finding all maximal-sum subsequences, and finding the longest segment with lower-bound average. These variations on the basic theme demonstrate the utilities of the techniques developed here.

The rest of the paper is organized as follows. Section 2 gives a formal definition of the RMSQ problem and introduces the RMQ techniques [5]. Section 3 considers a special case of the RMSQ problem (called the SRMSQ problem). Section 4 gives an optimal algorithm for the RMSQ problem. Section 5 uses the RMSQ techniques to solve three relevant problems in linear time.

2 Preliminaries

The input is a nonempty sequence $A = \langle a_1, a_2, \dots, a_n \rangle$ of real numbers. The maximum-sum segment of A is simply the contiguous subsequence having the greatest total sum. For simplicity, throughout the paper, the term “subsequence” will be taken to mean “contiguous subsequence”. To avoid ambiguity, we disallow nonempty, zero-sum prefix or suffix (also called ties) in the maximum-sum segments. For example, consider the input sequence $A = \langle 4, -5, 2, -2, 4, 3, -2, 6 \rangle$. The maximum-sum segment of A is $M = \langle 4, 3, -2, 6 \rangle$, with a total sum of 11. There is another subsequence tied for this sum by appending $\langle 2, -2 \rangle$ to the left end of M , but this subsequence is not the maximum-sum segment since it has a nonempty zero-sum prefix.

Definition 1. For an interval $[i, j]$, $1 \leq i \leq j \leq n$, let $A(i, j) = \langle a_i, \dots, a_j \rangle$ be the subsequence of A over the interval $[i, j]$. Let $S(i, j)$ be the maximum-sum segment of $A(i, j)$. Let $C[i]$ be the prefix sum of A over the interval $[1, i]$. Let $C[0] = 0$. Then $S(i, j) = C[j] - C[i - 1]$, $1 \leq i \leq j \leq n$.

2.1 Problem Definitions

We start with a special case of the RMSQ problem, called the SRMSQ problem.

A Special Case of the RMSQ problem (SRMSQ)

Input to be Preprocessed: A nonempty sequence of n real numbers $A = \langle a_1, a_2, \dots, a_n \rangle$.

Online Query: For an interval $[i, j]$, $1 \leq i \leq j \leq n$, $\text{SRMSQ}(A, i, j)$ returns a pair of indices (x, y) with $i \leq x \leq y \leq j$ such that $A(x, y)$ is the maximum-sum segment of $A(i, j)$.

A naïve algorithm is to build an $n \times n$ table storing the answers for all possible queries. Each entry (i, j) in the table represents a querying interval $[i, j]$. Since $i \leq j$, we only have to fill in the upper-triangular part of the table. By applying Bentley’s linear time algorithm for finding the maximum-sum segment of a sequence, we have an $O(n^3)$ -time preprocessing algorithm. Notice that answering an SRMSQ query now requires just one lookup to the table. To achieve

$O(n^2)$ -time preprocessing rather than the $O(n^3)$ -time naïve preprocessing, we take advantage of the online manner of the algorithm, filling in the table row-by-row. The total time required is therefore equivalent to the size of the table since each entry can be computed in constant time. In the paper, we give an algorithm that achieves $O(n)$ preprocessing time, and $O(1)$ time per query.

The Range Maximum-Sum Segment Query problem (RMSQ)

Input to be Preprocessed: A nonempty sequence of n real numbers $A = \langle a_1, a_2, \dots, a_n \rangle$.

Online Query: For two intervals $[i, j]$ and $[k, l]$, $1 \leq i \leq j \leq n$ and $1 \leq k \leq l \leq n$, $\text{RMSQ}(A, i, j, k, l)$ returns a pair of indices (x, y) with $i \leq x \leq j$ and $k \leq y \leq l$ that maximizes $S(x, y)$.

This is a generalized version of the SRMSQ problem because when $i = k$ and $j = l$, we are actually querying $\text{SRMSQ}(i, j)$. A naïve algorithm is to build a 4-dimensional table and the time for preprocessing is $\Omega(n^4)$. We give an algorithm that achieves $O(n)$ preprocessing time and $O(1)$ time per query for this problem.

2.2 The RMQ Techniques

Now we describe an important technique, called RMQ, used in our algorithm. We are given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$ to be preprocessed. A Range Minima Query (RMQ) specifies an interval $[i, j]$ and the goal is to find the index k with $i \leq k \leq j$ such that a_k achieves minimum.

Lemma 1. *Given a sequence A of n real numbers, we can preprocess A in $O(n)$ time so that any RMQ can be answered in $O(1)$ time.*

The well known algorithm for the RMQ problem is to first construct the Cartesian tree (defined by Vuillemin in 1980 [13]) of the sequence, then be preprocessed for LCA (Least Common Ancestor) queries [12, 7]. This algorithm can be easily modified to output the index k for which a_k achieves the minimum or the maximum. We let RMQ_{\min} denote the minimum query and RMQ_{\max} denote the maximum query. That is, $\text{RMQ}_{\min}(A, i, j)$ returns index k with $i \leq k \leq j$ such that a_k achieves the minimum, and $\text{RMQ}_{\max}(A, i, j)$ returns index k such that a_k achieves the maximum. For correctness of our algorithm if there are more than one minimum (maximum) in the querying interval, it always outputs the rightmost (leftmost) index k for which a_k achieves the minimum (maximum). This can be done by constructing the Cartesian tree in a particular order.

3 Coping with the SRMSQ Problem

The SRMSQ problem is to answer queries comprised of a single interval. Our strategy for preprocessing is to first find the “good partner” for each index of the sequence such that every index and its partner constitute a candidate solution for the maximum-sum segment. Then by applying RMQ techniques one can retrieve the pair with the greatest total sum within any given interval in constant time. Our first attempt for preprocessing is described as follows.

- Intuitively, one may pick index p with $1 \leq p \leq k$ that minimizes $C[p-1]$ as a partner of k , since the sum $S(p, k) = C[k] - C[p-1]$ will then be maximized. Record the partner of each index k in an array of length n , say $P[\cdot]$, and the sum $S(p, k)$ in an array of length n , say $M[\cdot]$.
- Apply RMQ_{\max} preprocessing to array $M[\cdot]$ for later retrieve.

It's not hard to see that the maximum-sum segment of subsequence $A(1, j)$ for all $1 \leq j \leq n$ can be retrieved by simply querying $\text{RMQ}_{\max}(M, 1, j)$. But when it comes to an arbitrary subsequence $A(i, j)$ for all $1 \leq i \leq j \leq n$, the partners we found might go beyond the left end of $[i, j]$. In this case, we have to "update" these partners since they no longer constitute candidate solutions for the maximum-sum segment of the subsequence $A(i, j)$. Such updates may cost linear per query in the worst case. Hence, the challenge now is to find a somehow "better" partner such that updates for arbitrary queries can be done in constant time.

Definition 2. $L[k] = A[1, k]$ for $1 \leq k \leq n$.
 $l \leftarrow 1 \leq l \leq k-1$ while $C[l] \geq C[k]$ do $l \leftarrow l-1$;
 $C[k] > C[l']$ for $1 \leq k' \leq k-1$ do $L[k] = 0$

Definition 3. $P[k] = A[1, k]$ for $1 \leq k \leq n$.
 $p \leftarrow L[k] + 1 \leq p \leq k$ while $C[p-1] < C[p]$ do $p \leftarrow p-1$

Definition 4. $A(P[k], k)$ for $1 \leq k \leq n$.
 $M[k] = S(P[k], k)$ for $1 \leq k \leq n$

Instead of finding the partner for each index k of A within the range $[1, k]$, we slightly narrow down the range from $[1, k]$ to $[L[k] + 1, k]$. The relationship between $L[\cdot]$ and $P[\cdot]$ defined above is illustrated in Fig. 1. The three arrays $C[\cdot]$, $P[\cdot]$, and $M[\cdot]$ can be computed by COMPUTE-CPM in Fig. 2.

Lemma 2. $C[\cdot], P[\cdot], M[\cdot] \in O(n)$

The correctness of the values of $C[\cdot]$ is trivial, and the values of $M[\cdot]$ are correct if the values of $P[\cdot]$ are correct. We first show by induction on k that the algorithm correctly computes array $L[\cdot]$. The basis, $k = 0$, is immediate. In the algorithm, $L[k]$ is the current working pointer scanning from right to left, searching for the largest index whose cumulative sum is greater than or equals to $C[k]$. Since by induction $L[1], L[2], \dots, L[k-1]$ have been computed correctly, the while-loop examines monotonically increasing values, $C[L[k]], C[L[L[k]]], \dots, C[L[\dots L[L[k]] \dots]]$, until it finally finds one or reaches 0.

Now we discuss the correctness of the values of $P[\cdot]$. By definition, $P[k]$ is the largest index lying in $[L[k] + 1, k]$ that minimizes $C[P[k] - 1]$. In each iteration of the while-loop, the range where $P[k]$ lies in, i.e. $[L[k] + 1, k]$, is about to be extended to $[L[L[k]] + 1, k]$. Since by induction $P[L[k]]$ is the largest index lying

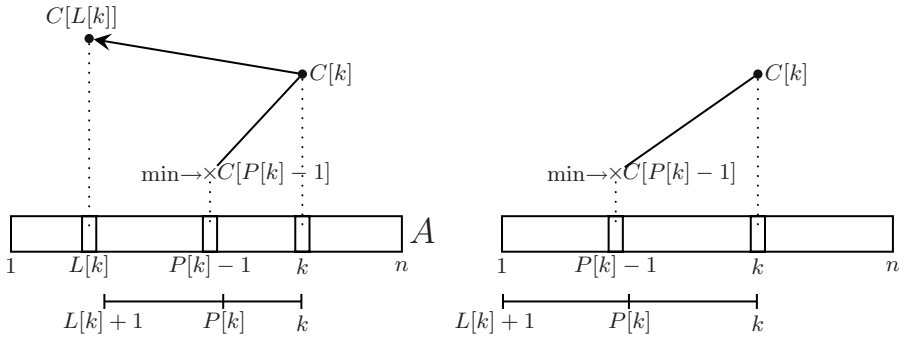


Fig. 1. An illustration for $L[\cdot]$ and $P[\cdot]$. Note that y -axis is the value $C[k]$ for the various k 's. The left side shows the case that there exists a largest index l with $1 \leq l \leq k - 1$ such that $C[l] \geq C[k]$. And the right side shows the case that $C[k]$ is the unique maximum of $C[k']$ for all $1 \leq k' \leq k - 1$

Algorithm COMPUTE-CPM

Input: An nonempty array of n real numbers $A[1 \dots n]$.

Output: An array $C[\cdot]$ of length $n + 1$ and two arrays $P[\cdot]$ and $M[\cdot]$ of length n .

```

1   $C[0] \leftarrow 0;$ 
2  for  $k \leftarrow 1$  to  $n$  do
3     $C[k] \leftarrow C[k - 1] + A[k];$ 
4     $L[k] \leftarrow k - 1;$    $P[k] \leftarrow k;$ 
5    while  $C[L[k]] < C[k]$  and  $L[k] > 0$  do
6      if  $C[P[L[k]] - 1] < C[P[k] - 1]$  then  $P[k] \leftarrow P[L[k]];$ 
7       $L[k] \leftarrow L[L[k]];$ 
8    end while
9     $M[k] \leftarrow C[k] - C[P[k] - 1];$ 
10 end for

```

Fig. 2. Algorithm for computing $C[\cdot]$, $P[\cdot]$, and $M[\cdot]$

in $[L[L[k]] + 1, L[k]]$ that minimizes $C[P[L[k]] - 1]$, the algorithm checks if $P[k]$ needs to be updated in line 6 to guarantee that $C[P[k] - 1]$ is minimized for the extended range. Hence, when the while-loop terminates both the values of $L[k]$ and $P[k]$ will be correctly computed.

The total number of operations of the algorithm is clearly bounded by $O(n)$ except for the while-loop body of lines 5-7. In the following, we show that the amortized cost of the while-loop is a constant. Let $\Phi(k)$ be the number of times $L[k]$ may advance forward before it reaches 0, i.e. $\Phi(k)$ is the minimal integer

$\overbrace{\Phi(k) \text{ times}}$

such that $\overbrace{L[\dots L[L[k]] \dots]} = 0$. In each iteration $\Phi(k)$ is increased by one and then possibly decreased a bit; however since $\Phi(k)$ can at most be increased by n in

total, and can never be negative, it cannot be decreased by more than n times. Thus the while loop is bounded by $O(n)$. \square

The following two lemmas show that each pair $(P[k], k)$ constitutes a candidate solution, i.e. $A(P[k], k)$, for the maximum-sum segment of A .

Lemma 3. *Let p and k be indices of A such that $p \leq k$. Then $A(p, k)$ is the maximum-sum segment of A if and only if $C[p - 1] \leq C[k']$ for all $L[k] \leq k' \leq k - 1$.*

Suppose not, then either p lies in $[1, L[k]]$ or p lies in $[L[k] + 1, k]$ but $C[p - 1]$ is not the rightmost minimum of $C[k']$ for all $L[k] \leq k' \leq k - 1$. We discuss both cases in the following.

1. Suppose index p lies in the interval $[1, L[k]]$. Then $S(p, L[k]) = C[L[k]] - C[p - 1] \geq C[k] - C[p - 1] = S(p, k)$. The equality must hold for otherwise $A(p, k)$ cannot be the maximum-sum segment of A . It follows $S(L[k] + 1, k) = C[k] - C[L[k]] = 0$. Hence $A(L[k] + 1, k)$ would be a zero-sum suffix of $A(p, k)$, which contradicts to the definition of the maximum-sum segment.
2. Suppose index p lies in the interval $[L[k] + 1, k]$. We know $C[p - 1]$ must be minimized for otherwise $A(p, k)$ cannot be the maximum-sum segment. If $C[p - 1]$ is not the rightmost minimum, i.e. there exists an index $k' > p$ such that $C[k' - 1] = C[p - 1]$ is also a minimum, then $S(p, k' - 1) = C[k' - 1] - C[p - 1] = 0$, which means $A(p, k)$ has a zero-sum prefix $A(p, k' - 1)$.

Hence, index p must be the largest index with $L[k] + 1 \leq p \leq k$ that minimizes $C[p - 1]$, i.e. $p = P[k]$. \square

Lemma 4. *Let r and k' be indices of A such that $1 \leq k' \leq n$. Then $A(P[r], r)$ is the maximum-sum segment of A if and only if $M[r] \geq M[k']$ for all $1 \leq k' \leq n$.*

Suppose on the contrary that segment $A(p, k)$ is the maximum-sum segment of A and $(p, k) \neq (P[r], r)$. By Lemma 3, we have $p = P[k]$. So

$$M[k] = S(P[k], k) = S(p, k) > S(P[r], r) = M[r]$$

which contradicts to $M[r]$ is the maximum value. \square

Therefore, once we have computed $M[\cdot]$ and $P[\cdot]$ for each index of A , to find the maximum-sum segment of A , we only have to retrieve index r such that $M[r]$ is the maximum value of $M[k']$ for all $1 \leq k' \leq n$. Then, candidate segment $A(P[r], r)$ is the maximum-sum segment of A .

Lemma 5. *Let k be an index of A such that $P[k] < k$. Then $C[P[k] - 1] < C[k'] < C[k]$ for all $P[k] \leq k' \leq k - 1$.*

Suppose not. That is, there exists an index k'' which lies in $[P[k], k - 1]$ such that $C[k''] \leq C[P[k] - 1]$ or $C[k''] \geq C[k]$. By the definition of $P[k]$, we know that $C[k''] \leq C[P[k] - 1]$ cannot hold. If $C[k''] \geq C[k]$, then again by the definition of $P[k]$, we know $P[k]$ must lie in $[k'' + 1, k]$. Thus, $k'' < P[k] \leq k''$. A contradiction occurs. \square

In other words, $C[P[k] - 1]$ is the unique minimum of $C[k']$ for all $P[k] - 1 \leq k' \leq k$ and $C[k]$ is the unique maximum. The following key lemma shows the nesting property of the candidate segments. (See Fig. 4 for an illustration of the nesting property. This important property makes “update in constant time” possible as we will show in later proof.)

Lemma 6. *Suppose $k \leq l$, $k < l$, $P[k] < P[l] \leq k < l$.*

Suppose $P[k] < P[l] \leq k < l$ holds. By Lemma 5, we have $C[P[k] - 1] < C[k'] < C[k]$ for all $P[k] \leq k' \leq k - 1$ and $C[P[l] - 1] < C[k''] < C[l]$ for all $P[l] \leq k'' \leq l - 1$. Since the two intervals $[P[k] - 1, k]$ and $[P[l] - 1, l]$ overlap, it's not hard to see that $C[P[k] - 1] < C[k'''] < C[l]$ for all $P[k] \leq k''' \leq l - 1$. It follows that $L[l] < P[k] - 1$. Thus, $C[P[k] - 1] < C[P[l] - 1]$ with $L[l] + 1 \leq P[k] \leq l$ is a contradiction to that $C[P[l] - 1]$ is minimized with $L[l] + 1 \leq P[l] \leq l$. \square

Now, we are about to establish the relationship between sequence A and its subsequence $A(i, j)$. The following lemma shows that some good partners of A , say $P[s]$, do not need to be “updated” for the subsequence $A(i, j)$ if $[P[s], s]$ doesn't go beyond $[i, j]$.

Lemma 7. *Suppose $s, i \leq P[s] \leq s \leq j, P[s] \leq s$ and $A(i, j)$.*

Let $C^*[k]$ be the cumulative sum of $A(i, j)$. Then we have $C^*[k] = C[k] - C[i - 1]$ for $i - 1 \leq k \leq j$. Let $L^*[k]$ be the left bound of $A(i, j)$ at index k for $i \leq k \leq j$ and $P^*[k]$ be the good partner of $A(i, j)$ at index k .

If $L[s] \geq i$, we have $L^*[s] = L[s]$ since $L[s]$ is the largest index l with $i \leq l \leq s - 1$ such that $C^*[L[s]] = C[L[s]] - C[i - 1] \geq C[s] - C[i - 1] = C^*[s]$. Otherwise, i.e. $L[s] < i$, we have $L^*[s] = i - 1$. Therefore we can conclude that $L[s] \leq L^*[s]$. Moreover, since minimizing $C[P[s] - 1]$ minimizes $C^*[P[s] - 1] = C[P[s] - 1] - C[i - 1]$, it's not hard to see that $P[s]$ is still the largest index with $L^*[s] + 1 \leq P[s] \leq s$ that minimizes $C^*[P[s] - 1]$, i.e. $P^*[s] = P[s]$. \square

Corollary 1. *Suppose $s, i \leq P[s] \leq s \leq j, M[s] \leq s$ and $A(i, j)$.*

A direct result of Lemma 7. \square

Now, we are ready to present our main algorithm for the SRMSQ problem, which is given in Fig. 3. As an example, in Fig. 4, the input sequence A has 15 elements. Suppose we are querying $\text{SRMSQ}(A, 3, 7)$. QUERY OF SRMSQ in Fig. 3 first retrieves index r such that $M[r]$ is maximized with $3 \leq r \leq 7$ (line 1). In this case, $r = 5$, which means candidate segment $A(P[5], 5)$ has the largest sum compared with other candidate segments whose ending indices lie in $[3, 7]$. Since $A(P[5], 5)$ doesn't go beyond interval $[3, 7]$, the algorithm outputs $(P[5], 5)$, which means segment $A(3, 5)$ is the maximum-sum segment of the subsequence $A(3, 7)$.

Suppose we are querying $SRMSQ(A, 6, 12)$. Since $[P[9], 9]$ goes beyond the left end of $[6, 12]$, lines 3-9 are executed. In line 3, $RMQ_{min}(c, 5, 8)$ retrieves index 8. In line 4, $RMQ_{max}(m, 10, 12)$ retrieves index 11. In line 5, since $C[9] - C[8] = 6 < M[11] = 8$, the algorithm outputs $(P[11], 11)$, which means $A(11, 11)$ is the maximum-sum segment of the subsequence $A(6, 12)$.

Algorithm PREPROCESS OF $SRMSQ(A)$

- 1 Run COMPUTE-CPM to compute $C[\cdot], P[\cdot]$, and $M[\cdot]$ of A .
- 2 Apply RMQ_{min} preprocessing to array $C[\cdot]$.
- 3 Apply RMQ_{max} preprocessing to array $M[\cdot]$.

Algorithm QUERY OF $SRMSQ(A, i, j)$

- 1 $r \leftarrow RMQ_{max}(M, i, j)$
- 2 **if** $P[r] < i$ **then**
- 3 $p \leftarrow RMQ_{min}(C, i - 1, r - 1) + 1$;
- 4 $s \leftarrow RMQ_{max}(M, r + 1, j)$;
- 5 **if** $C[r] - C[p - 1] < M[s]$ **then**
- 6 OUTPUT $(P[s], s)$;
- 7 **else**
- 8 OUTPUT (p, r) ;
- 9 **end if**
- 10 **else**
- 11 OUTPUT $(P[r], r)$;
- 12 **end if**

Fig. 3. Algorithm for the SRMSQ problem

Lemma 8. Let C^*, P^*, M^* be the cumulative sum, the good partner, and the sum of candidate segment of $A(i, j)$ at index k for $i \leq k \leq j$, respectively.

Let r be the index satisfying $M[r] \geq M[k]$ for all $i \leq k \leq j$ (line 1).

1. If $i \leq P[r] \leq r \leq j$ (lines 10-11): Our goal is to show that $M^*[r] \geq M^*[k]$ for all $i \leq k \leq j$, and then by Lemma 4 $A(P[r], r)$ is the maximum-sum segment of $A(i, j)$.
 - (a) First, we consider each index k' where $i \leq P[k'] \leq k' \leq j$. By corollary 1, we have $M^*[k'] = M[k'] \leq M[r] = M^*[r]$.
 - (b) Next, we consider each index k'' where $P[k''] < i \leq k'' \leq j$. By Lemma 5 we have $C[P[k''] - 1] < C[k] < C[k'']$ for all $P[k''] \leq k \leq k'' - 1$. Since by definition $P^*[k''] - 1$ must lie in $[i - 1, k'' - 1]$, we have $C[P[k''] - 1] < C[P^*[k''] - 1]$. Hence, we can deduce that $M^*[k''] = C^*[k''] - C^*[P^*[k''] - 1] = C[k''] - C[P^*[k''] - 1] < C[k''] - C[P[k''] - 1] = M[k''] \leq M[r] = M^*[r]$.

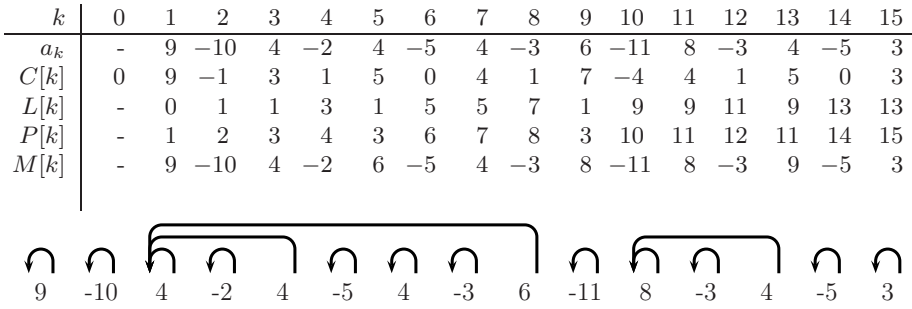


Fig. 4. The candidate segment $A(P[k], k)$ of A at each index k . Notice that, the pointer at each index k points to the position of $P[k]$

Thus, by (a) and (b) we conclude that $M^*[r] \geq M^*[k]$ for all $i \leq k \leq j$.

2. If $P[r] < i \leq r \leq j$ (lines 2-9):

(a) First, we consider each index k' where $i \leq k' \leq r - 1$. By Lemma 5, we have $C[P[r] - 1] < C[k] < C[r]$ for all $P[r] \leq k \leq r - 1$. For any index p' with $i \leq p' \leq k'$, since $S(p', k') = C[k'] - C[p'] < C[r] - C[p'] = S(p', r)$, k' cannot be the right end of the maximum-sum segment of $A(i, j)$.

(b) Next, we consider each index k'' where $r + 1 \leq k'' \leq j$. By Lemma 6, we know that it cannot be the case $P[r] < P[k''] \leq r < k''$. Suppose $P[k''] \leq P[r] < r < k''$, then by Lemma 5 we have $C[P[k''] - 1] < C[r] < C[k'']$ and $C[P[k''] - 1] \leq C[P[r] - 1] < C[k'']$. It follows that $M[k''] = C[k''] - C[P[k''] - 1] > C[r] - C[P[r] - 1] = M[r]$ which contradicts to that $M[r] \geq M[k]$ for all $i \leq k \leq j$. Thus, it must be the case $r < P[k''] \leq k'' \leq j$. Therefore by Corollary 1 we have $M^*[k''] = M[k'']$.

Let p be the largest index that minimizes $C[p - 1]$ with $i \leq p \leq r$ (line 3). Let index s satisfy $M[s] \geq M[k]$ for all $r + 1 \leq k \leq j$ (line 4). One can easily deduce by (a), (b), and Lemma 4 that either $A(p, r)$ or $A(P[s], s)$ is the maximum-sum segment of $A(i, j)$ (lines 5-9). □

As you can see, if $A(P[r], r)$ does not go beyond i then we are done. Otherwise, it turns out that for each index k' which lies in $[i, r - 1]$, k' cannot be the right end of the maximum-sum segment of $A(i, j)$. On the other hand, for each index k'' which lies in $[r + 1, j]$, the good partner of k'' doesn't need to be updated. Hence, only the good partner of index r needs to be updated. The time required for each query can therefore achieve constant. We summarize our main result in the following theorem.

Theorem 1. *Let A be an array of integers. For any query (i, j) , the maximum-sum segment of $A(i, j)$ can be found in $O(n)$ time, where n is the length of A .*

4 Coping with the RMSQ Problem

The RMSQ problem is to answer queries comprised of two intervals $[i, j]$ and $[k, l]$, where $[i, j]$ specifies the range of the starting index of the maximum-sum

segment, and $[k, l]$ specifies the range of the ending index. It is meaningless if the range of the starting index is in front of the range of the ending index, and vice versa. Therefore we assume, without loss of generality, that $i \leq k$ and $j \leq l$. We give our main result of the RMSQ problem as follows.

Theorem 2. *Let A be an array of n numbers. Then the maximum-sum segment query can be answered in $O(n)$ time.*

We discuss it under two possible conditions.

1. Nonoverlapping case ($j \leq k$): Suppose the intervals $[i, j]$ and $[k, l]$ do not overlap. Since $S(x, y) = C[y] - C[x - 1]$, maximizing $S(x, y)$ is equivalent to maximizing $C[y]$ and minimizing $C[x - 1]$ with $i \leq x \leq j$ and $k \leq y \leq l$. By applying RMQ techniques to preprocess $C[\cdot]$, the maximum-sum segment can be located by simply querying $\text{RMQ}_{\min}(C, i - 1, j - 1)$ and $\text{RMQ}_{\max}(C, k, l)$.
2. Overlapping case ($j > k$): When it comes to the overlapping case, just to retrieve the maximum cumulative sum and the minimum cumulative sum might go wrong if the minimum is on the right of the maximum in the overlapping region. There are three possible cases for the maximum-sum segment $A(x, y)$.
 - (a) Suppose $i \leq x \leq k$ and $k \leq y \leq l$, which is a nonoverlapping case. To maximize $S(x, y)$, we retrieve the minimum cumulative sum by querying $\text{RMQ}_{\min}(C, i - 1, k - 1)$ and the maximum cumulative sum by querying $\text{RMQ}_{\max}(C, k, l)$.
 - (b) Suppose $k + 1 \leq x \leq j$ and $j \leq y \leq l$. This is also a nonoverlapping case.
 - (c) Otherwise, i.e. $k + 1 \leq x \leq j$ and $k + 1 \leq y \leq j$. This is exactly the same as an $\text{SRMSQ}(A, k + 1, j)$ query.

The maximum-sum segment $A(x, y)$ must be the one of these three possible cases which has the greatest sum. □

5 Solving Three Relevant Problems in Linear Time

Given a sequence of n numbers, a lower bound L , and an upper bound U , the first problem is to find the maximum-sum segment of length at least L and at most U [10, 4]. It's not hard to see that it suffices to find for each index $k \geq L$ the maximum-sum segment whose starting index lies in $[\max(1, k - U + 1), k - L + 1]$ and ending index is k . Since our RMSQ algorithm can answer each such query in $O(1)$ time, the total running time is therefore linear.

The second problem is to find those nonoverlapping, contiguous subsequences having greatest total sum. The highest maximal-sum subsequence is simply the maximum-sum segment of the sequence. The k^{th} maximal-sum subsequence is defined to be the maximum-sum segment disjoint from the $k - 1$ maximal-sum subsequences. Additionally, we stop the process when the next best sum is nonpositive. By applying Bentley's linear time algorithm, all maximal-sum subsequences can be found in $O(n^2)$ time in the worst case. Ruzzo and Tompa [11] proposed a genius linear time algorithm for this problem. In the paper, our SRMSQ techniques immediately suggest an alternative divide-and-conquer

algorithm for finding all maximal-sum subsequences in linear time: query the maximum-sum segment of the sequence, remove it, and then apply the SRMSQ query recursively to the left of the removed portion, and then to the right. Since our SRMSQ algorithm can answer each such query in $O(1)$ time, the total running time is therefore linear.

The third problem is to identify the longest segment with lower-bound average [14]. Given a sequence of n numbers and a lower bound N , the goal is to find the longest consecutive subsequence with the average value of those numbers in the subsequence at least N . First, we obtain a new sequence of n numbers $B = \langle b_1, b_2, \dots, b_n \rangle$ by subtracting N from each of the numbers in the given sequence. It's not hard to see that it suffices to find the longest segment in B with nonnegative sum. Our strategy is to compute for each index k of B the longest nonnegative segment starting at position k . The key idea is that if $A(k, l)$ has nonnegative sum, then by appending the maximum-sum segment starting at index $l + 1$ and ending in $[l + 1, n]$ to the right end of $A(k, l)$ we get a longer segment $A(k, l')$. Again, our RMSQ techniques can achieve this in constant time. We repeat the process until the sum of the next extended segment is negative. In this way we can find $A(k, R[k])$, the longest segment starting at k with nonnegative sum. Another trick to achieve linear running time is the key observation proposed by Wang and Xu in [14]. Let k and l be the current working pointer scanning from left to right. Assume that we have computed $R[k_1]$ for some index k_1 . For each index k_2 lying in $[k_1 + 1, R[k_1]]$, if $C[k_2] \geq C[k_1]$ then we have $R[k_2] \leq R[k_1]$ since otherwise we would get a longer nonnegative segment starting at k_1 by appending $A(R[k_1] + 1, R[k_2])$ to $A(k_1, R[k_1])$. Therefore we can bypass the computation of k_2 and move k forward from k_1 until $C[k] < C[k_1]$. Next we can move l forward from $\max(R[k_1] + 1, k)$ by appending the maximum-sum segment, since $A(k, R[k_1])$ certainly has nonnegative sum when $C[k] < C[k_1]$ with $k < R[k_1]$. The procedure terminates when k or l reaches the end of B . It is clear that the total running time is linear since either the value of k or l increases by at least one in each iteration, and never decreases.

Acknowledgments. We thank Yu-Ru Huang, Rung-Ren Lin, Hsueh-I Lu, and An-Chiang Chu for helpful conversations. Kuan-Yu Chen and Kun-Mao Chao were supported in part by an NSC grant 92-2213-E-002-059.

References

1. M. A. Bender, and M. Farach-Colton. The LCA Problem Revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, 17: 88–94, 2000.
2. J. Bentley. *Programming Pearls - Algorithm Design Techniques*, CACM, 865–871, 1984.
3. K. Chung and H.-I. Lu. An Optimal Algorithm for the Maximum-Density Segment Problem. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, LNCS 2832, 136–147, 2003.
4. T.-H. Fan, S. Lee, H.-I. Lu, T.-S. Tsou, T.-C. Wang, and A. Yao. An Optimal Algorithm for Maximum-Sum Segment and Its Application in Bioinformatics. *CIAA*, LNCS 2759, 251–257, 2003.

5. H. Gabow, J. Bentley, and R. Tarjan. Scaling and Related Techniques for Geometry Problems. *Proc. Symp Theory of Computing*(STOC), 135–143, 1984.
6. D. Gusfield. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1999.
7. D. Harel and R. E. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J Comput.*, 13: 338–355, 1984.
8. X. Huang. An Algorithm for Identifying Regions of a DNA Sequence that Satisfy a Content Requirement. *CABIOS*, 10: 219–225, 1994.
9. Y.-L. Lin, X. Huang, T. Jiang, and K.-M. Chao, MAVG: Locating Non-Overlapping Maximum Average Segments in a Given Sequence, *Bioinformatics*, 19:151-152, 2003.
10. Y.-L. Lin, T. Jiang, and K.-M. Chao. Efficient Algorithms for Locating the Length-constrained Heaviest Segments with Applications to Biomolecular Sequence Analysis. *Journal of Computer and System Sciences*, 65: 570–586, 2002.
11. W. L. Ruzzo and M. Tompa. A Linear Time Algorithm for Finding All Maximal Scoring Subsequences. In *7th Intl. Conf. Intelligent Systems for Molecular Biology, Heidelberg, Germany*, 234–241, 1999.
12. B. Schieber and U. Vishkin. On Finding Lowest Common Ancestors: Simplification and Parallelization. *SIAM J Comput.*, 17: 1253–1262, 1988.
13. J. Vuillemin. A Unifying Look at Data Structures. *CACM*, 23: 229–239, 1980.
14. L. Wang and Y. Xu. SEGID:Identifying Interesting Segments in (Multiple) Sequence Alignments. *Bioinformatics*, 19: 297–298, 2003.

An Efficient Algorithm for Finding Maximum Cycle Packings in Reducible Flow Graphs

Xujin Chen and Wenan Zang^{*,**}

Department of Mathematics, The University of Hong Kong,
Hong Kong, China
wzang@maths.hku.hk

Abstract. Reducible flow graphs occur naturally in connection with flow-charts of computer programs and are used extensively for code optimization and global data flow analysis. In this paper we present an $O(n^2 m \log(n^2/m))$ algorithm for finding a maximum cycle packing in any weighted reducible flow graph with n vertices and m arcs.

Keywords: feedback set, cycle packing, network flow, algorithm, complexity.

1 Introduction

Let G be a digraph with a nonnegative integral weight $w(e)$ (resp. $w(v)$) on each arc e (resp. vertex v). A collection \mathcal{C} of cycles (repetition is allowed) of G is called a w -cycle packing if each arc e (resp. vertex v) of G is used at most $w(e)$ (resp. $w(v)$) times by members of \mathcal{C} ; a set X of arcs (resp. vertices) in G is called a w -feedback set (resp. w -vertex feedback set) if $G - X$ contains no cycle. The w -cycle packing problem is to find a cycle packing with maximum size, and the w -feedback set problem consists in finding a feedback set with minimum total weight. These two problems clearly form a primal-dual pair in integer programming and thus are closely tied to each other. While the latter is a well-known NP -hard problem [13] and has been studied extensively, the present paper concerns itself with the former, which also arises in a variety of applications. Recently, Caprara, Panconesi, and Rizzi [7] gave a thorough and detailed analysis of the hardness and approximability of the cycle packing problem on undirected graphs. We point out that a slight modification of their approaches can lead to essentially the same statements for the problem on digraphs; that is, it admits no fully polynomial time approximation scheme unless $P = NP$, and can be approximated within a factor of $\frac{1}{2 \log n}$, where n is the number of vertices in the input digraph and the base of \log is 2. We focus our study of the cycle packing problem on reducible flow graphs in this paper.

Reducible flow graphs (or simply reducible graphs) occur naturally in connection with flow-charts of computer programs and are used extensively for code

* Corresponding author.

** Supported in part by the Research Grants Council of Hong Kong.

optimization and global data flow analysis, so they have attracted tremendous research efforts in the past three decades. Hopcroft and Ullman¹ obtained the first efficient algorithm for recognizing reducible flow graphs, which was improved by Tarjan [23]. The reader is referred to Hecht and Ullman [16] for good characterizations of all reducible flow graphs. In [21], Shamir gave a linear time algorithm for finding minimum feedback vertex sets in reducible graphs, and an $O(n^2m \log(n^2/m))$ algorithm was discovered by Ramachandran [18] for finding a minimum weighted feedback arc set in weighted reducible graphs and for finding a minimum weighted feedback vertex set in vertex-weighted reducible graphs with n vertices and m arcs. In [19], Ramachandran proved that the cardinality of a minimum feedback arc set in a reducible flow graph is equal to the cardinality of a maximum collection of arc disjoint cycles, thereby establishing a conjecture of Frank and Gyarfás [12]; her proof also leads to an $O(m^2)$ algorithm for finding the corresponding set of arc disjoint cycles. We remark that Ramachandran's method [19] can be further extended to find maximum weighted cycle packings in weighted reducible flow graphs. One subroutine of this approach, called $O(n)$ times, is a maximum flow algorithm, in which the so-called newly added arcs must be saturated by the flows at each step, so the augmenting path method for the maximum flow has to be applied; this leads to a gap between the time complexity of the algorithm for the cycle packing problem and that for feedback arc set problem in weighted reducible flow graphs as, to the best of our knowledge, none of the most efficient maximum flow algorithms currently known for general networks is based on this method directly. The purpose of this paper is to bridge this complexity gap and present an $O(nf(n, m) + m^2)$ algorithm for finding a maximum cycle packing in any weighted reducible flow graph, where $f(n, m)$ is the optimal time complexity of an algorithm for finding a maximum flow in any flow network with n vertices and m arcs, which is at most $O(nm \log(n^2/m))$. Our algorithm assembles several ideas introduced by Ramachandran [18, 19] and heavily relies on the laminar structure of reducible flow graphs. Moreover, it can use any fastest (integral) maximum flow algorithm as its subroutine.

The remainder of this paper is organized as follows. In section 2, we give some preliminary results on reducible flow graphs and network flows. In section 3, we present the algorithm and establish its correctness. In section 4, we conclude this paper with some remarks and open problems.

2 Preliminaries

Let us now introduce some notions and terminologies. Let $G = (V, A)$ be a digraph. We denote by (u, v) an arc in A from its tail u to its head v . A walk in G is a finite sequence $W = v_0e_1v_1 \dots e_kv_k$, whose terms are alternately vertices and arcs, such that $e_i = (v_{i-1}, v_i)$ for $1 \leq i \leq k$. If v_0, v_1, \dots, v_k are distinct, W

¹ in: *Proc. 6th Annual Princeton Conference on Information Sciences and Systems*, Princeton, NJ, 1972, pp. 119-122.

is called a $v_0 - v_k$ or a $v_0 - v_k$; if v_0, v_1, \dots, v_{k-1} are distinct, $v_k = v_0$, and $k \geq 2$, W is called a $v_0 - v_k$. We call G if G contains no cycles. For convenience, we let $P[u, v]$ denote the section of a path P from u to v , and set $P(u, v) = P[u, v] \setminus \{u\}$ and $P[u, v) = P[u, v] \setminus \{v\}$.

A or is a digraph G with a distinguished vertex r , called its , such that there is a path in G from r to every vertex in G . Let $G = (V, A, r)$ be a flow graph, and let $u, v \in V$. We say that u (or u is a of v) if every path $r - v$ path in G passes through u . Let R be a subgraph of G . A vertex v in R is called an of R if $v = r$ or if there is an arc (u, v) of G with u outside R . A of G is a maximal acyclic subgraph of G rooted at r . A of G is a DAG containing a directed spanning tree with root r grown by the depth first search (DFS) algorithm [22, 24]. Graph G is called if the depth first search DAG of G is unique. Unless G is acyclic, DFS also discovers , which are arcs of G not included in the depth first search DAG.

The following characterizations of reducible flow graphs will be used repeatedly in this paper.

Theorem 2.1. $G = (V, A, r)$ $D = (V, A_D, r)$ $B = A \setminus A_D$

- (i) G
- (ii) G F (FIG. 1)
- (iii) D G
- (iv) A G A_1 A_2 (V, A_1, r) G u v G (v, u) A_2
- (v) C G $e \in B$ C C

The equivalence of (i)-(iv) can be found in [16, 17]. The implication (i) \Rightarrow (v) is contained in [21], and the converse (v) \Rightarrow (i) follows from (ii) and the fact that no vertex of the cycle in F dominates all vertices in this cycle. \square

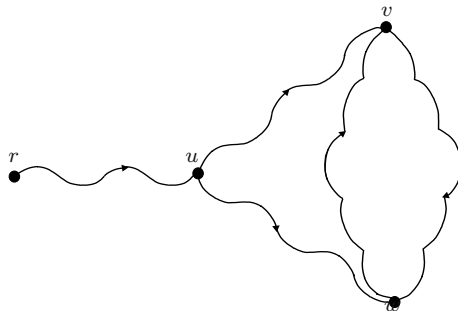


Fig. 1. The Forbidden Subgraph F

As shown by Aho and Ullman [3], the dominator relation in a flow graph $G = (V, A, r)$ can be represented in the form of a tree rooted at r . Let V_h denote the set of heads of all back arcs in G . The Head Dominator Tree T_h of G , defined by Ramachandran [18], represents the domination relation restricted to $V_h \cup \{r\}$. For each $u \in V_h \cup \{r\}$, its Region $R(u)$ is defined to be the subgraph of G induced by all vertices that are dominated by u (see FIG. 2). Clearly, $R(r) = G$.

The following lemma will play a crucial role in the design and analysis of our algorithm.

Lemma 2.2. Let $G = (V, A, r)$ be a flow graph and let $u, v \in V_h \cup \{r\}$. Then

- (i) $R(u) \cap R(v) = R(u \vee v)$
- (ii) $u \in R(v) \iff R(u) \subseteq R(v)$
- (iii) $R(u) \cap R(v) = R(u) \iff R(u) \subseteq R(v)$
- (iv) $R(u) \cap R(v) = R(v) \iff R(v) \subseteq R(u)$

Proof. Omitted. □

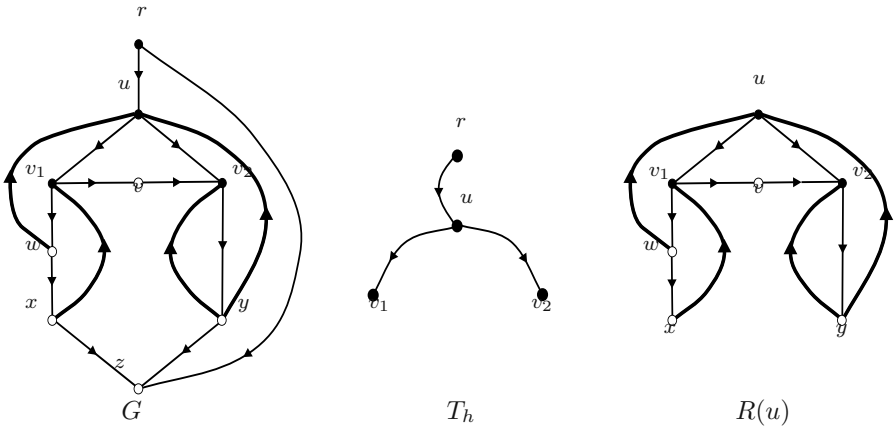


Fig. 2. The Head Dominator Tree T_h and Region $R(u)$

Lemma 2.3. Let $G = (V, A, r)$ be a flow graph and let $u, v \in V_h \cup \{r\}$. Then $\pi(u) < \pi(v)$ if and only if $R(u) \cap R(v) = R(u)$. Here $\pi(u) = |R(u) \cap V_h|$, $n = |V|$, and $m = |A|$.

Proof. Omitted. □

To solve the maximum cycle packing problem on reducible flow graphs, Ramachandran [18, 19] made extensive use of network flow techniques, and we shall

apply her basic ideas and develop the techniques. As usual, a $s-t$ flow in a digraph $N = (V, A, s, t, c)$ is a digraph $G = (V, A)$ with two distinguished vertices, a source s and a sink t , and a nonnegative integral flow $x(u, v)$ on each arc (u, v) . See Ahuja, Magnanti and Orlin [1] and Ford and Fulkerson [11] for in-depth accounts of network flows. It is clear that any subgraph (V', A') of G with $\{s, t\} \subseteq V'$ corresponds to a digraph $N' = (V', A', s, t, c)$ of N . For convenience, we let $|x|$ denote the value of a flow x .

Lemma 2.4. *Let $N' = (V', A', s, t, c)$ be a subgraph of $N = (V, A, s, t, c)$. Let μ and μ' be the maximum $s-t$ flow values in N and N' , respectively. Let λ and λ' be the maximum $s-t$ flow values in N and N' with the constraint $\lambda' \leq \lambda \leq \mu$. Then $|x| = \lambda \geq \sum_{(u,t) \in A'} x(u, t) \geq \lambda'$, where $n = |V|$ and $m = |A|$.*

Let us construct a network $\bar{N} = (\bar{V}, \bar{A}, s, q, c)$ from N as follows: first add two new vertices p and q and two new arcs (p, q) and (t, q) with capacities $c(p, q) = \lambda'$ and $c(t, q) = \lambda - \lambda'$, then subdivide each arc (u, t) with $u \in V'$ into two arcs (u, \bar{u}) , (\bar{u}, t) and add one arc (\bar{u}, p) such that $c(u, \bar{u}) = c(u, t)$ and $c(\bar{u}, t) = \infty = c(\bar{u}, p)$. The construction of \bar{N} is complete. In \bar{N} , vertex s remains to be the source while q becomes the sink. We claim that the maximum $s-q$ flow value of \bar{N} is λ .

To justify the claim, by the max-flow min-cut theorem we may turn to prove that the capacity of a minimum $s-q$ cut of \bar{N} is λ . Since $[\bar{V} \setminus \{q\}, \{q\}]$ is clearly an $s-q$ cut of \bar{N} with capacity λ , it suffices to prove that any $s-q$ cut $[U, \bar{U}]$ in \bar{N} is of capacity at least λ , where $s \in U$ and $q \in \bar{U}$. To this end, let us distinguish among the following three cases. If $t \in \bar{U}$, then from the construction of \bar{N} we deduce that either $[U, \bar{U}]$ is of capacity ∞ or $[U, \bar{U}]$ corresponds to an $s-t$ cut of N whose capacity is at least μ ($\geq \lambda$) by the max-flow min-cut theorem; if $t \in U$ and $p \in \bar{U}$, then $[U, \bar{U}]$ contains arc (t, q) with capacity $c(t, q) = \lambda - \lambda'$, and $[U, \bar{U}] \setminus \{(t, q)\}$ corresponds to an $s-t$ cut in N' with capacity at least μ' ($\geq \lambda'$); if $\{t, p\} \subseteq U$, then both (p, q) and (t, q) are contained in $[U, \bar{U}]$. So in all three cases the capacity of $[U, \bar{U}]$ is at least λ , as claimed.

Now let \bar{x} be a maximum $s-q$ flow in \bar{N} . It follows from the construction of \bar{N} that \bar{x} corresponds to an $s-t$ flow x of N with the desired properties. Since a maximum flow in \bar{N} can be found [14] in time $O(|\bar{V}||\bar{A}| \log(|\bar{V}|^2/|\bar{A}|)) = O(nm \log(n^2/m))$, we are done. \square

To establish the correctness of our algorithm, we need to decompose a flow in a network into flows on paths. So the following theorem (see [1]) will be applied.

Theorem 2.5 (Flow Decomposition Theorem). *Let x be a flow of value k in a digraph $N = (V, A, s, t, c)$. Then x can be decomposed into ℓ flows P_1, P_2, \dots, P_ℓ such that $\sum_{i=1}^\ell y_i = k$, where y_i is the value of flow P_i .*

- $\ell \leq m$,
- $\sum_{i=1}^\ell y_i = k$,

$$\bullet \sum_{i: (u,v) \in P_i} y_i = x(u,v) \quad \dots \quad (u,v) \in A$$

$$\bullet n = |V| \quad \dots \quad m = |A| \quad \square$$

For simplicity, we denote the above ... of x by $\{y_1 P_1, y_2 P_2, \dots, y_\ell P_\ell\}$.

3 Algorithm

Let $G = (V, A, r)$ be a reducible flow graph with a nonnegative integral weight $w(e)$ on each arc $e \in A$, and let V_h, T_h , and $R(u)$ be as defined in the preceding section. In [18, 19], Ramachandran transformed the cycle packing problem on a reducible flow graph into a maximum flow problem. Let us present Ramachandran's construction of the corresponding flow network. For each $u \in V_h \cup \{r\}$, let $G(u)$ denote the network obtained from $R(u)$ by first splitting each head v in $R(u) \cap V_h$ into two vertices v and v' and then adding a new vertex t . Each DAG arc entering (resp. leaving) the original head v in G corresponds to one that enters (resp. leaves) the newly formed head v , and each back arc entering the original v corresponds to one that enters v' in $G(u)$. Moreover, there is an arc (v', t) with infinity capacity. The capacity of any other arc in $G(u)$ is equal to its corresponding weight in G . We propose to call v' the ... of v .

Let us now inductively construct a new network $N(u)$ for each $u \in V_h \cup \{r\}$ as follows²: If u is a vertex with no child in T_h , set $N(u) = G(u)$; else, let v_1, v_2, \dots, v_k be all the children of u on T_h , and let the maximum flow value of $N(v_i)$ be c_i for $1 \leq i \leq k$. Then the vertex set of $N(u)$ is the same as that of $G(u)$, and the arc set of $N(u)$ is the union of all arcs in $G(u)$, $N(v_i)$ for $i = 1, 2, \dots, k$, and $\{(u, v_i) : i = 1, 2, \dots, k\}$. The capacity of each arc (u, v_i) is set to be c_i .

For $G = (V, A, r)$ in FIG. 2, the construction of $G(u)$ and $N(u)$ is illustrated in FIG. 3.

Using the laminar structure of reducible flow graphs, we shall get an improved algorithm for finding a maximum cycle packing in any arc-weighted reducible graph.

Lemma 3.1. ... $\pi(a) < \pi(b)$

- (i) $a' \dots a) \dots N(b) \dots b' \dots N(a)$
- (ii) $\dots a \dots b \dots N(r)$
- (iii) $\dots N(a) \setminus \{t\} \dots N(b) \setminus \{t\} \dots N(b) \dots N(a)$

(i) follows instantly from the construction of $N(r)$; (ii) can be seen from Lemma 2.3; and (iii) can be deduced from Lemma 2.2(iv), (i) and the construction of $N(r)$. □

² We note that $G(u)$ is essentially $G_m(u)$ in [18], and $N(u)$ is very similar to $G_{mm}(u)$.

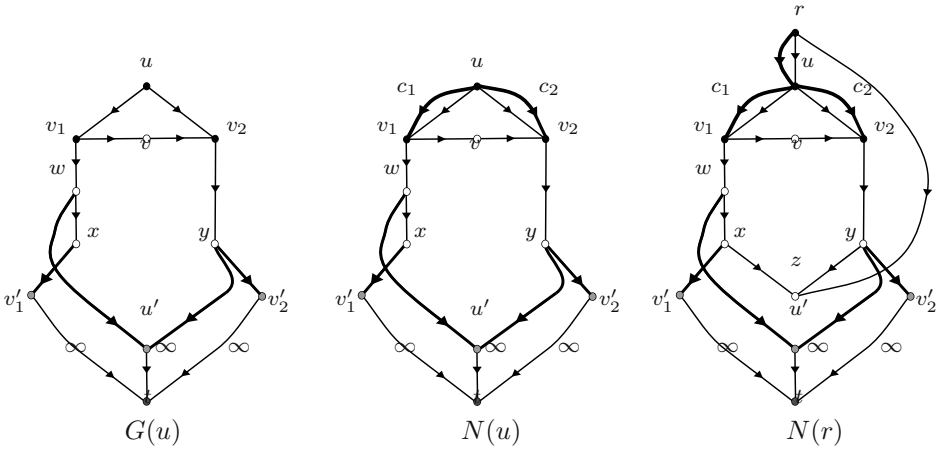


Fig. 3. The Construction of $G(u)$ and $N(u)$

From the above construction we see that $N(u)$ is obtained from $G(u)$ by adding all arcs in the subtree of T_h rooted at u . An arc of $N(u)$ is called *newly added* if it is outside $G(u)$. An $r - t$ flow x in $N(r)$ is called *good* if $x(u, v) \leq \sum_{a' \in N(v)} x(a', t)$ holds for any newly added arc (u, v) of $N(r)$.

Lemma 3.2. *Let x be an arbitrary integral maximum $r - t$ flow in $N(r)$. Then there exists a newly added arc (u, v) of $N(r)$ such that $x(u, v) > \sum_{a' \in N(v)} x(a', t)$. Here $n = |V|$ and $m = |A|$.*

Let x be an arbitrary integral maximum $r - t$ flow in $N(r)$. We may assume that x is not good and so

- (1) some newly added arc (p, q) of $N(r)$ satisfies

$$x(p, q) > \sum_{a' \in N(q)} x(a', t).$$

Let π be the DFS order of T_h exhibited in Lemma 2.3, and let (u, v) be a newly added arc among all those (p, q) described in (1) such that

- (2) $\pi(v)$ is minimized, that is, $\pi(v) \leq \pi(q)$ for any above-mentioned arc (p, q) .

Notice that $N(r)$ is acyclic, so, by Theorem 2.5, x admits a path flow decomposition $\{y_1 P_1, y_2 P_2, \dots, y_\ell P_\ell\}$. Without loss of generality, we assume that $P_1, P_2, \dots, P_\alpha$ are all the paths in this decomposition that passes through v . Using flow conservation, we get

- (3) $\sum_{j=1}^\alpha y_j \geq x(u, v)$ for the above newly added arc (u, v) .

Set $Q_j = P_j[v, t]$ (the subpath of P_j from v to t) for $1 \leq j \leq \alpha$. It follows from Lemma 2.2(ii), the definition of $N(v)$, and Theorem 2.5 that

- (4) for each arc (a, b) of $N(v)$, we have $x(a, b) = \sum_{j=1: (a,b) \in Q_j}^\alpha y_j$.

We extract from $N(r)$ a flow network $N'(v)$ with source v , sink t , and capacity function c' as follows: $N'(v)$ is the union of $N(v)$ and all Q_j for $j = 1, 2, \dots, \alpha$, and the capacity

(5) $c'(a, b)$ is set to be $c(a, b)$ if (a, b) is an arc in $N(v)$ and to be $\sum_{j=1}^{\alpha} y_j$ if (a, b) is in $N'(v) \setminus N(v)$.

Since $\{y_1 Q_1, y_2 Q_2, \dots, y_{\alpha} Q_{\alpha}\}$ is a path flow decomposition of a $v - t$ flow in $N'(v)$ of value $\sum_{i=1}^{\alpha} y_i$, and $x(u, v)$ is no more than the maximum flow value of network $N(v)$ (recall the construction of $N(r)$), (3), (5), and Lemma 2.4 (with $N(v)$, $N'(v)$, $\sum_{j=1}^{\alpha} y_j$ and $x(u, v)$ in place of N' , N , λ and λ' over there, respectively) guarantee the existence of an integral $v - t$ flow z in $N'(v)$ such that

$$(6) \sum_{j=1}^{\alpha} y_j = |z| \geq \sum_{a' \in N(v)} z(a', t) \geq x(u, v).$$

Define an integral vector x' on the arcs of $N(r)$ by

(7) $x'(a, b) = x(a, b)$ if (a, b) is outside $N'(v)$ and $x'(a, b) = z(a, b) + x(a, b) - \sum_{j=1}^{\alpha} y_j$ otherwise.

It is easy to see from (5) and (7) that

(8) x' is also an integral maximum $r - t$ flow in $N(r)$.

(9) $x'(a', t) \leq x(a', t)$ for any vertex $a' \in N'(v) \setminus N(v)$.

Indeed, since (a', t) is contained in $N'(v)$ for $a' \in N'(v) \setminus N(v)$, from (7) we deduce that $x'(a', t) = z(a', t) + x(a', t) - \sum_{j=1}^{\alpha} y_j \leq c'(a', t) + x(a', t) - \sum_{j=1}^{\alpha} y_j = x(a', t)$ by (5), as desired.

We propose to prove that

(10) for any newly added arc (p, q) of $N(r)$ with $\pi(q) \leq \pi(v)$,

$$x'(p, q) \leq \sum_{a' \in N(q)} x'(a', t).$$

Suppose the contrary: the above inequality is violated by some newly added arc (p, q) with $\pi(q) \leq \pi(v)$. Since the newly added arcs form the edge set of the head dominator tree T_h , (u, v) is the unique newly added arc entering v . By (6) and (7), we have $\pi(q) < \pi(v)$. Note that $N'(v)$ is a digraph rooted at v , by Lemma 3.1(ii), $N'(v)$ does not contain any $a \in V_h \cup \{r\}$ with $\pi(a) < \pi(v)$. So $x'(p, q) = x(p, q) \leq \sum_{a' \in G(q)} x(a', t)$, implying $\sum_{a' \in G(q)} x'(a', t) < \sum_{a' \in G(q)} x(a', t)$. Hence, $N(q)$ and $N'(v)$ must have some vertex b other than t in common. We claim that $N(v) \subseteq N(q)$. For this purpose, by Lemma 3.1(iii), suppose the contrary: $N(q) \cap N(v) = \{t\}$. Considering a path in $N'(v) - \{t\}$ from v to b , we deduce from Lemma 3.1(ii) that this path contains an entry vertex of $N(q) \setminus \{t\}$ other than q . However, it follows from Lemma 2.2(ii), (iii) and the construction of $N(q)$ that q is the unique entry vertex of $N(q) \setminus \{t\}$, a contradiction. So the claim is justified. Thus,

$$\begin{aligned} 0 &> \sum_{a' \in N(q)} x'(a', t) - \sum_{a' \in N(q)} x(a', t) = \sum_{a' \in N(q) \cap N'(v)} [x'(a', t) - x(a', t)] \\ &= \sum_{a' \in N(v)} [x'(a', t) - x(a', t)] + \sum_{a' \in N(q) \cap (N'(v) \setminus N(v))} [x'(a', t) - x(a', t)] \quad (\text{by claim}) \\ &\geq \sum_{a' \in N(v)} [x'(a', t) - x(a', t)] + \sum_{a' \in N'(v) \setminus N(v)} [x'(a', t) - x(a', t)] \quad (\text{by (9)}) \end{aligned}$$

$$\begin{aligned}
 &= \sum_{a' \in N'(v)} x'(a', t) - \sum_{a' \in N'(v)} x(a', t) \\
 &= \sum_{a' \in N'(v)} \left[z(a', t) + x(a', t) - \sum_{j=1}^{\alpha} y_j \right] - \sum_{a' \in N'(v)} x(a', t) \quad (\text{by (7)}) \\
 &= |z| - \sum_{j=1}^{\alpha} y_j = 0 \quad (\text{by (6)}),
 \end{aligned}$$

this contradiction completes the proof of (10).

Let us replace x by x' and repeat the process. From (2), (8) and (10) we conclude that a good integral maximum $r - t$ flow in $N(r)$ can be obtained after at most n iterations. Since the initial maximum flow x and x' in (7) can both be found in $O(nm \log(n^2/m))$ time [14], the whole algorithm runs in time $O(n^2m \log(n^2/m))$. \square

Lemma 3.3. *Let x be a flow in $N(r)$ with value $|x| = m = |A|$. Then there exists a good flow x' in $N(r)$ with value $|x'| = m$ and x' is subject to (i), (ii).*

The statement clearly holds if x is a zero flow. So we assume x is nonzero. Let $p(u)$ denote the parent of u on T_h for each $u \in V_h \setminus \{r\}$, let π be the DFS order of T_h specified in Lemma 2.3, and let v' , the image of v , be the vertex of $N(r)$ (recall the construction) such that

- (i) $x(v', t) > 0$, and
- (ii) subject to (i), $\pi(v)$ is maximized.

Condition (i) guarantees the existence of an $r - t$ path P through v' in $N(r)$ such that $x(e) > 0$ for any arc e on P . Now let $Q = u_0u_1 \dots u_k$ denote the path from r to v on T_h , where $u_0 = r$, $u_k = v$, and $u_i = p(u_{i+1})$ for $i = 0, 1, \dots, k - 1$. By Lemma 2.2(ii), (iii), and the construction of $N(r)$, u_i is the only entry vertex of $N(u_i) \setminus \{t\}$ for $0 \leq i \leq k$. Hence vertices u_0, u_1, \dots, u_k appear sequentially on P .

Using Lemma 3.1(i), we have

- (1) v' is contained in no $N(u)$ with $\pi(u) > \pi(v)$, and v' is contained in $N(u)$ with $\pi(u) < \pi(v)$ iff u dominates v .

We claim that

- (2) $P[v, v']$ contains no arc $(p(u), u)$ of T_h with $\pi(p(u)) \geq \pi(v)$.

Suppose to the contrary that such u exists. Since $x(p(u), u) > 0$ and x is a good flow, $x(a', t) > 0$ for some vertex a' in $N(u)$. Thus $\pi(a) \geq \pi(u) > \pi(v)$, contradicting the selection (ii) of v' .

Now let R be the path obtained from P by replacing $P[u_i, u_{i+1}]$ with the newly added arc (u_i, u_{i+1}) whenever $x(u_i, u_{i+1}) > 0$ for $i = 1, 2, \dots, k - 1$. Then $x(e) > 0$ for each arc e on R . Let δ denote the minimum $x(e)$ on R . We define a vector x' on the arc set of $N(r)$ as follows: $x'(e) = x(e) - \delta$ if e is an arc on R and $x(e)$ otherwise. From (1) and (2), we can conclude that x' remains to be a good flow of $N(r)$. Let e' denote the arc of P entering v' , let C be the cycle

of G uniquely contained in the union of $P[v, v']$ and the back arc corresponding to e' , and let \mathcal{C} contain C such that the multiplicity of C is δ . Replace x by x' and repeat the process until x becomes a zero flow. Clearly, \mathcal{C} is a cycle packing with size equal to the value of the initial $r - t$ flow.

Since P can be found by breadth first search in $O(m)$ time and there are at most m iterations according to the definition of C , the algorithm runs in $O(m^2)$ time. \square

Lemma 3.4. Let \mathcal{C} be a cycle packing of G such that $k \leq m$, $\sum_{i=1}^k y_i = r - t$, and $\sum_{i=1}^k |x| = \sum_{i=1}^k y_i$.

Let C_1, C_2, \dots, C_k be all cycles in \mathcal{C} such that the multiplicity of each C_i is y_i . Recall Theorem 2.1(v), each C_i contains precisely one back arc (v_i, u_i) . Let P_i be the unique path from r to u_i on T_h , and let Q_i denote the concatenation of $P_i, C_i \setminus \{(v_i, u_i)\}, (v_i, u'_i)$, and (u'_i, t) for $i = 1, 2, \dots, k$. We aim to show that $\{y_1 Q_1, y_2 Q_2, \dots, y_k Q_k\}$ is a path flow decomposition of a flow x in $N(r)$. For this purpose, it suffices to check the capacity constraint.

Suppose to the contrary that the capacity constraint is violated on some arc (u, v) . Then (u, v) must be a newly added arc. We select such an arc so that $\pi(v)$ is maximized. Thus

- (i) $\sum_{i: (u,v) \in P_i} y_i > c(u, v)$ and
- (ii) $\sum_{i: (a,b) \in P_i} y_i \leq c(a, b)$ for all arcs (a, b) of $N(v)$.

Among Q_1, Q_2, \dots, Q_k , we see from their construction that only those, say $Q_1, Q_2, \dots, Q_\alpha$, with $\alpha \leq k$, that contain a vertex in $N(v) \setminus \{v\}$ can pass through (u, v) . It follows from Theorem 2.1(v) and the definition of $N(v)$ that $Q_i[v, t]$ is entirely contained in $N(v)$ for each i . In view of (ii), $\{y_1 Q_1, y_2 Q_2, \dots, y_\alpha Q_\alpha\}$ is a path flow decomposition of a $v - t$ flow in $N(v)$. Observe that $\sum_{i=1}^\alpha y_i$ is bounded above by the maximum flow value of $N(v)$ which is $c(u, v)$ by the construction of $N(r)$. Hence, $\sum_{i: (u,v) \in Q_i} y_i = \sum_{i=1}^\alpha y_i \leq c(u, v)$, contradicting (i). This completes the proof. \square

We are ready to present our algorithm for finding a maximum cycle packing in any arc-weighted reducible flow graph.

Algorithm for Finding a Maximum Cycle Packing

- Input: An arc-weighted reducible flow graph $G = (V, A, r)$.
- Output: A maximum cycle packing \mathcal{C} of G .

Step 0. Construct the flow network $N(r)$ as defined at the beginning of this section.

Step 1. Find a good integral maximum $r - t$ flow x in $N(r)$ as described in Lemma 3.2.

Step 2. Convert x to a cycle packing \mathcal{C} of G as described in Lemma 3.3 and return \mathcal{C} .

The correctness of this algorithm follows instantly from Lemma 3.2-3.4. Since the dominator tree can be constructed in linear time [15], so can be the under-

lying digraph of $N(r)$. By calling the maximum flow algorithm [14] at most n times, we can get the capacities of all newly added arcs in $N(r)$. In view of the complexity stated in each lemma, we conclude that the algorithm runs in $O(n^2m \log(n^2/m))$ time.

4 Concluding Remarks

In this paper we have obtained a polynomial time algorithm for finding a maximum weighted cycle packing in any arc-weighted reducible flow graph. We remark that our algorithm can also be employed to solve the maximum cycle packing problem in the vertex weighted case as the latter can be easily transformed into the former. Furthermore, there is a linear time algorithm for finding maximum vertex-disjoint cycles in any reducible flow graph (the dual of Shamir's problem [21]).

Ramachandran [19] contains a beautiful minimax arc theorem for reducible flow graphs. Although her theorem is concerning the unweighted case, her proof implicitly yields a stronger statement; that is, in any arc-weighted reducible flow graph, the maximum size of a cycle packing is equal to the minimum total weight of a feedback arc set, which also implies the minimax relation for the vertex-weighted case (this unweighted case is due to Frank and Gyafas [12]). Minimax relations play important roles in combinatorics and optimization. In addition to their great theoretical interest, they often yield polynomial-time solutions of the corresponding problems. Major open problems in this direction are to characterize all digraphs with the minimax arc (resp. vertex) relation on packing and covering cycles, for any nonnegative integral weight function defined on the arc (resp. vertex) set. See Cai et al. [4, 5, 6] for a complete characterization of all tournaments and bipartite tournaments and Ding et al. [8, 9] for the description of all undirected graphs.

Ramachandran [20] came up with parallel algorithms for recognizing reducible flow graphs, for finding dominators, and for finding a minimum feedback vertex set in an unweighted reducible flow graph. Certainly, parallel algorithms for the general cycle packing and feedback set problems on reducible flow graphs also deserve good research efforts.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows – Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, New Jersey (1993)
2. Ahuja, R.K., Orlin, J.B., Tarjan, R.E.: Improved time bounds for the maximum flow problem. *SIAM J. Comput.* **18** (1989) 939-954
3. Aho, A.V., Ullman, J.D.: Principle of Compiler Design. Addison-Wesley Reading, MA (1977)
4. Cai, M.C., Deng, X.T., Zang, W.: A TDI system and its application to approximation algorithms. in: Proc. 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA (1998) pp. 227-233

5. Cai, M.C., Deng, X.T., Zang, W.: An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.* **30** (2001) 1993-2007
6. Cai, M.C., Deng, X.T., Zang, W.: A min-max theorem on feedback vertex sets. *Math. Oper. Res.* **27** (2002) 361-371
7. Caprara, A., Panconesi, A., Rizzi, R.: Packing cycles in undirected graphs. *J. Algorithms* **48** (2003) 239-256
8. Ding, G., Zang, W.: Packing cycles in graphs. *J. Combin. Theory Ser. B* **86** (2002) 381-407
9. Ding, G., Xu, Z., Zang, W.: Packing cycles in graphs, II. *J. Combin. Theory Ser. B* **87** (2003) 244-253
10. Dinits, E.A.: Algorithms for solution of a problem of maximum flow in a network with power estimation. *Soviet Mathematics Doklady* **11** (1970) 1277-1280
11. Ford, L.R. Jr., Fulkerson, D.R.: *Flows in Networks*. Princeton Univ. Press, Princeton, NJ (1962)
12. Frank, A., Gyarfás, A.: Directed graphs and computer programs. in: *Problemes Combinatoires et Theorie des Graphes, Colloque Internationaux C.N.R.S.* **260** (1976) pp. 157-158
13. Garey, M.R., Johnson, D.S.: *Computers and Intractability*. W.H. Freeman and Company, New York (1979)
14. Goldberg, A. and Tarjan, R.E.: A new approach to the maximum flow problem. in: *Proc. 18th Annu. ACM Symp. on Theory of Computing* (1985) pp. 136-146
15. Harel, D.: A linear algorithm for finding dominators in flow graphs and related problems. in: *Proc. 17th Annual ACM Symp. on Theory of Computing* (1984) pp. 185-194
16. Hecht, M.S., Ullman, J.D.: Flow graph reducibility. *SIAM J. Comput.* **1** (1972) 188-202
17. Hecht, M. S., Ullman, J.D.: Characterizations of reducible graphs. *J. Assoc. Comput. Mach.* **21** (1974) 367-375
18. Ramachandran, V.: Finding a minimum feedback arc set in reducible flow graphs. *J. Algorithms* **9** (1988) 299-313
19. Ramachandran, V.: A minimax arc theorem for reducible flow graphs. *SIAM J. Discrete Math.* **3** (1990) 554-560
20. Ramachandran, V.: Parallel algorithms for reducible flow graphs. *J. Algorithms* **23** (1997) 1-31
21. Shamir, A.: A linear time algorithm for finding minimum cutsets in reducible graphs. *SIAM J. Comput.* **8** (1979) 645-655
22. Tarjan, R.E.: Depth-first search and linear graph algorithm. *SIAM J. Comput.* **1** (1972) 146-160
23. Tarjan, R.E.: Testing flow reducibility. *J. Comput. System Sci.* **9** (1974) 355-365
24. Tarjan, R.E.: *Data Structure and Network Algorithms*. SIAM, Philadelphia, PA, (1983)

Efficient Job Scheduling Algorithms with Multi-type Contentions^{*}

Zhenming Chen, Vikas Singh, and Jinhui Xu

Department of Computer Science and Engineering,
State University of New York at Buffalo,
Buffalo, NY 14260, USA
{zchen4, vsingh, jinhui}@cse.buffalo.edu

Abstract. In this paper, we consider an interesting generalization of the classic job scheduling problem in which each job needs to compete for not only machines but also other types of resources. The contentions among jobs for machines and resources could interfere with each other, which complicates the problem dramatically. We present a family of approximation algorithms for solving several variants of the problem by using a generic algorithmic framework. Our algorithms achieve a constant approximation ratio (i.e., 3) if there is only one type of resources or certain dependency relation exists among multiple types of resources. For the case that r unrelated resources are given, the approximation ratio of our algorithm becomes $k + 2$, where $k \leq r$ is a constant depending on the problem instance. As an application, we also show that our techniques can be easily applied to optical burst switching (OBS) networks for deriving more efficient wavelength scheduling algorithms.

1 Introduction

Job scheduling problem is a fundamental problem in theoretical computer science and finds numerous applications in many different areas. Extensive research [1, 2, 3, 5, 6, 7, 8, 9, 10, 14] has been done on different variants of this problem. One commonly used model of this problem is the so called job interval selection (JIS) problem in which a set of jobs, J_1, J_2, \dots, J_n , with each $J_i, 1 \leq i \leq n$, associated with a profit and a set of intervals representing its possible schedulings, and a set of machines M_1, M_2, \dots, M_m are given, and the objective is to schedule a subset of the jobs on the m machines so that the total profit of the scheduled jobs is maximized. Such a model captures the essence of many scheduling problems, and has led to a number of efficient solutions [5, 7, 10]. In general, this problem is NP-hard, and its hardness mainly comes from the contentions among jobs while competing for machines.

In some applications, it is quite common that jobs need to compete not only for machines, but also for other critical (or exclusive) resources. Some jobs may

^{*} This research was supported in part by an IBM faculty partnership award, and an IRCAF award from SUNY Buffalo.

need to obtain other resources before becoming executable on machines. For example, in optical burst switching (OBS) networks[11], a burst (or packet) can either be scheduled directly on a wavelength (or channel) if there is one available at the arrival time of the burst or be delayed by a fiber delay line (FDL) so that the burst can be scheduled in a different time slot. In such applications, bursts need to compete for both wavelengths (which can be viewed as machines) and FDLs. The multiple types of contentions among jobs are in general quite different and cannot be handled similarly. They could interfere with each other and dramatically change the nature of the scheduling problem.

To develop efficient techniques for solving such problems, we consider a generalization of the job interval selection problem called *Job Interval Scheduling with Multiple Contentions* (JISMC). In the JISMC problem, besides the sets of jobs and machines, we are also given a set of resources for the jobs to compete for. Each job may require a subset of the resources and the resources may change some parameters associated with the job, such as profit, starting and ending time. Dependency relation may exist among resources, and a dependency graph can be derived. Different dependency graphs need different scheduling algorithms. In this paper, we first give a general model of this problem and then generalize the technique developed in [1, 3, 4] for solving the JIS problem to derive approximation algorithms for our JISMC problem. We consider three types of dependency relations among resources and give approximation algorithm for each of them. More specifically, we present a 3-approximation algorithm for the JISMC problem in which there is only one type of resources. In the case that $r (\geq 1)$ types of resources exist, we show that a 3-approximation is possible if certain dependency relation exists among all the resources. For the most general case in which all r types of resources are unrelated, our algorithm achieves a $(k+2)$ -approximation, where $k \leq r$ is a constant depending on problem instances.

Due to the space limit, lots of details and proofs are omitted from this extended abstract.

2 Problem Description

In the JISMC problem, the input is a set of n jobs $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$, a set of m machines $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$, and a set of r resources $\mathcal{R} = \{R_1, R_2, \dots, R_r\}$, the objective is to schedule a subset of jobs on the machines so that the total profit of the scheduled jobs is maximized. Each job J_i , $1 \leq i \leq n$, is associated with a positive profit w_i , a set of starting times $S_i = \{s_{i1}, s_{i2}, \dots, s_{ic_i}\}$, and a set of durations $L_i = \{l_{i1}, l_{i2}, \dots, l_{im}\}$ on each of the m machines. S_i and L_i together define a set of $\mathcal{I}_i^{M_j} = \{(s_{i1}, s_{i1} + l_{ij}), (s_{i2}, s_{i2} + l_{ij}), \dots, (s_{ic_i}, s_{ic_i} + l_{ij})\}$ for all possible schedulings of job J_i on machine M_j . Each job J_i may also request a subset of resources $\mathcal{R}_i \subseteq \mathcal{R}$ and \mathcal{R}_i could change the profit, starting time, and/or duration of J_i by a $\mathbf{v}_i = \{\Delta_i^w, \Delta_i^s, \Delta_i^l\}$. Corresponding to each machine interval $I_{ijk} = (s_{ik}, s_{ik} + l_{ij})$, $1 \leq j \leq m, 1 \leq k \leq c_i$, and each requested resource $R_l \in \mathcal{R}_i$, there is a $I_{ijk}^{R_l}$ indicating the period of I_{ijk}

using R_l . In general, the reservation interval should be the same as the corresponding machine interval. But a resource such as an FDL in OBS could change the starting time of a job and therefore make the two intervals different.

In the JISMC problem, we assume that the schedulings on both machines and resources are non-preemptive and the resources are exclusive (i.e., no two jobs can access a resource simultaneously). When the machines are identical, the machine intervals on all machines will be the same.

A resource R_j depends on another resource R_l if for each job $J_i \in \mathcal{J}$, $R_j \in \mathcal{R}_i$ implies $R_l \in \mathcal{R}_i$ and the reservation intervals of (every interval of) J_i on R_j and R_l are the same. Based on this relation, a dependency graph $G_{\mathcal{R}}$ can be constructed among all resources by adding a directed edge from each resource to every resource depending on it. For a set of resources, the dependency graph could be an arbitrary graph.

Our techniques for solving the JISMC problem are generalized from an interesting and powerful technique developed by Berman and DasGupta in [3, 4] for solving the JIS and related problems. Their technique contains two main phases:

Evaluation: All intervals are evaluated in a non-decreasing order of their ending times. Based on its contention with other intervals, each interval is given a value to indicate its potential profit. A subset of “profitable” intervals are kept for the next phase.

Selection: A subset of intervals is selected from the kept intervals to avoid any possible contention.

Their technique achieves a 2-approximation for the JIS problem with m non-identical machines and an $\frac{(m+1)^m}{(m+1)^m - m^m}$ -approximation for m identical machines. The success of their algorithm relies on a key fact that the contentions among intervals can be sorted by the ending time of the intervals and resolved by using such an order. This is because in the JIS problem there is only one kind of contention: jobs competing for machines. But in our JISMC problem, more than one type of contention exists, and different types of contentions may not be consistent with each other, making it much more difficult to apply their technique to our problem.

Our main idea for the JISMC problem is to “emulate” the two-phase algorithm in such a way that contentions on resources can be made consistent with that of the machines. We illustrate our ideas by first showing how to solve the scheduling problem with single resource and then extending it to the problem of multiple resources.

3 Scheduling Algorithm for Single Resource

In this section, we consider the case in which only one type of optional resource R is available. Resource R has $r \geq 1$ copies, R_1, R_2, \dots, R_r , each of them is independent from the others. We first consider the case in which the m machines are identical. The cases of non-optional resources and non-identical machines can be handled with minor modifications.

In this problem, each job J_i has c_i intervals, which can be scheduled on any of the m machines, with or without any of the r copies of R . Thus a job can be executed in a total of $c_i \times m \times (r + 1)$ different ways. To capture all possible schedulings and contentions for each job J_i , we first form a family of intervals $\mathcal{I}_i^M = \{I_{i1}, I_{i2}, \dots, I_{ic_i}\}$ for J_i according to its starting times and duration. The set of machine intervals represents all possible schedulings of J_i without using the resource R . If J_i requests R as its resource, then for each interval $I_{ij} \in \mathcal{I}_i^M$, we make r copies of I_{ij} by using I_{ij} and the modification vector \mathbf{v}_i associated with J_i . Each resource interval corresponds to a particular copy of the resource R and defines a possible scheduling of J_i on a machine. Let \mathcal{I}_i^R denote the set of resource intervals of J_i and $\mathcal{I}_i = \mathcal{I}_i^M \cup \mathcal{I}_i^R$ denote the family of intervals of job J_i .

Clearly, each job needs to compete with others for both machines and resources. To resolve the two types of contentions, ideally we should handle them separately. But the interference between them could be rather complicated and make it very difficult to achieve quality guaranteed solutions. To simplify the problem, our main idea is to establish a linear order based on one type of contention (e.g., the contention on machines) and use it as the primary order to schedule jobs. We also build proper data structures to maintain a secondary linear order for the other type of contention. Whenever there is a contention that needs to be resolved, we use either the primary, the secondary, or both orders.

Below are the main steps of our algorithm.

- Evaluation:**
1. Sort all intervals in $\bigcup_{i=1}^n (\mathcal{I}_i)$ in a non-decreasing order of the ending times.
 2. Initialize a stack $S_{M_j}, 1 \leq j \leq m$, for each machine M_j and a stack $S_{R_k}, 1 \leq k \leq r$, for each copy R_k of the resource R .
 3. Scan all intervals according to the sorted order, and for each interval $I \in J_i$ and each machine M_j ,
 - find the set $C(I)$ of intervals in the stacks $S_{M_j}, 1 \leq j \leq m$ which conflict with I and partition them into three subsets of intervals $C_f(I), C_{m_j}(I)$ and $C_r(I)$, where
 - (a) $C_f(I)$ is the set of intervals which are from the same job of I and pushed into any stack of the m machines before evaluating I ,
 - (b) $C_{m_j}(I)$ is the set of intervals from different jobs which are in S_{M_j} and overlap with I on time,
 - (c) $C_r(I)$ is the set of intervals which compete with I on the resource R_k in stack S_{R_k} , where R_k is the resource corresponding to I . If I is not a resource interval, then $C_r(I)$ is an empty set.
 - Compute the potential of I on machine M_j as follows.

$$I.value(j) = w_i + \Delta_i^w - \sum_{I' \in C_f(I) \cup C_{m_j}(I) \cup C_r(I)} I'.value,$$

where Δ_i^w is the change of profit due to the use of resource R . If I is not a resource interval, then $\Delta_i^w = 0$.

4. Compute the value of I as $I.value = \max_{j=1}^m \{I.value(j)\}$.
5. If $I.value > 0$, the interval I is pushed into the stack S_{M_p} , where M_p is the machine which has the maximum potential $I.value(p)$ for I among all machines. If I is a resource interval, then I is also pushed into the corresponding stack S_{R_k} .
6. Repeat the above procedure until all intervals are done.

- Selection:**
1. Mark all intervals in the m stacks of machines as \dots .
 2. In the reverse order of ending time, keep popping from one of the m stacks of machines the interval I of the largest ending time (among all intervals in the m stacks).
 3. If I is free, then assign the corresponding job J_i of I to the machine M_j whose stack contains I , and assign the resource R_k to J_i , where R_k is the resource corresponding to the interval I .
 4. Mark all intervals in $C_f(I), C_{m_j}(I)$ and $C_r(I)$ as \dots, \dots .
 5. If there are other intervals $\notin C_r(I)$ conflicting with I on R_k , mark them as occupied.
 6. Repeat the above procedure until no interval is in any stack.

Next, we will show that our algorithm generates a feasible scheduling which is a 3-approximation of the optimal solution under some easily satisfied conditions.

Let $\mathcal{S} = \{S_{M_1}, S_{M_2}, \dots, S_{M_m}\}$ be the set of stacks of the m machines after the evaluation phase, and C_f, C_m, C_r be defined as above. Let $V(X)$ be the sum of the values of all intervals in the set X , and A be any feasible scheduling. Let $C_F = \cup_{I \in A} C_f(I)$, $C_M = \cup_{I \in A} C_{m_j}(I)$, $C_R = \cup_{I \in A} C_r(I)$, where M_j is the machine scheduled to execute I by A . Let $P(A)$ be the sum of the profits of all jobs in A .

Definition 1. $\dots R \dots M \dots I_1 \dots I_2 \dots$
 $\dots R \dots R \dots M \dots$
 $\dots I_1 \dots I_2 \dots () \dots$
 $(\dots) \dots I_1 \dots I_2 \dots R \dots$
 $\dots M () I_1 \dots I_2 \dots M$
 $\dots R \dots R \dots M \dots$

Notice that in our JISMC problem, although a resource might be inconsistent with machines as both the reservation intervals and the modification vectors of jobs could be different for different intervals, it is quite unusual for that to happen in practice. This is because in practice most of the resources are either used jointly with the machine or used immediately before or after the machine. Thus the reservation intervals of jobs are either exactly the same as their corresponding intervals on machines or have the same order. This implies that they are consistent with the machines.

Definition 2. $\dots \dots \dots$
 $\dots \dots \dots$
 $\dots \dots \dots$
 $\dots \dots \dots$

Lemma 1. R

For each interval $I \in J_i$ selected during the selection phase, we associate it with the set of intervals in $C_f(I) \cup C_{m_j}(I) \cup C_r(I)$. Since I is pushed into the stack of machine M_j during the evaluation phase, $I.value > 0$. Thus, $w_i + \Delta_i^w \geq V(C_f(I) \cup C_{m_j}(I) \cup C_r(I)) + I.value$. The selection phase considers every interval in the non-increasing order of the ending times of intervals. If an interval I' is not marked as occupied, it will be included as part of the solution. By the consistency assumption, we know that the order of the set of inserted intervals in the stacks of resources will be the same as those in the stacks of machines. Thus an interval can be marked as occupied only if it is in $C_f(I) \cup C_{m_j}(I) \cup C_r(I)$ for some selected interval I . Hence every interval in the stacks of machines will be associated with at least one selected interval, and the lemma follows.

Lemma 2. $I_1 \quad I_2 \quad A$
 $C_{m_{j_1}}(I_1) \quad C_{m_{j_2}}(I_2), C_r(I_1) \quad C_r(I_2) \quad C_f(I_1) \quad C_f(I_2),$
 $M_{j_1} \quad M_{j_2}$
 $I_1 \quad I_2 \quad A$

Since I_1 and I_2 are in a feasible solution, they must belong to different jobs, thus $C_f(I_1) \cap C_f(I_2) = \phi$. If M_{j_1} and M_{j_2} are different machines, then $C_{m_{j_1}}(I_1) \cap C_{m_{j_2}}(I_2) = \phi$. Otherwise, since I_1 and I_2 are from a feasible scheduling, they do not overlap on time, thus $C_{m_{j_1}}(I_1) \cap C_{m_{j_2}}(I_2) = \phi$. If I_1 and I_2 are using different copies of R , clearly $C_r(I_1) \cap C_r(I_2) = \phi$. Otherwise, by the consistency assumption, the reservation intervals of I_1 and I_2 on R are disjoint, therefore $C_r(I_1) \cap C_r(I_2) = \phi$.

Lemma 3. R $V(S) + V(C_F) + V(C_R) \geq P(A)$

By Lemma 2, it is sufficient to prove that for each interval $I \in A$ of job J_i , $V(C_f(I)) + V(C_{m_j}(I)) + V(C_r(I)) \geq w_i + \Delta_i^w$, where M_j is the machine scheduled by A to execute I .

When evaluating I , since the value of interval I comes from the machine M_{min} whose $V(C_{m_{min}}(I))$ is the smallest, we have $V(C_{m_j}(I)) \geq V(C_{m_{min}})$. Thus it is sufficient to prove that $V(C_f(I)) + V(C_{m_{min}}(I)) + V(C_r(I)) \geq w_i + \Delta_i^w$.

Let S' be the set of intervals pushed into the stacks of the m machines before evaluating interval I . If $I.value \leq 0$, then we have $w_i + \Delta_i^w - V(C_f(I)) - V(C_{m_{min}}(I)) - V(C_r(I)) \leq 0$. Otherwise, I will be pushed into the stack $S_{M_{min}}$. Let S be $S' \cup \{I\}$. Let $V_1 = V(C_f(I))$ before pushing I into the stack and $V_2 = V(C_f(I))$ after pushing I . We have $V_2 \geq V_1 + I.value = V_1 + (w_i + \Delta_i^w) - V_1 - V(C_{m_{min}}(I)) - V(C_r(I)) = w_i + \Delta_i^w - V(C_{m_{min}}(I)) - V(C_r(I))$. Thus, $V(C_f(I)) + V(C_{m_{min}}(I)) + V(C_r(I)) \geq w_i + \Delta_i^w$.

Theorem 1. R

Let V be the total profit of the obtained scheduling, and A be an optimal solution. By Lemma 1, $V \geq V(\mathcal{S})$, and it is also easy to see that $V(C_F) \leq V(\mathcal{S})$ and $V(C_R) \leq V(\mathcal{S})$. Thus $3V \geq V(\mathcal{S}) + V(C_F) + V(C_R) \geq P(A)$.

Lemma 4.

The running time of the above algorithm is given by the following theorem, due to the space limit, the proof is left for the full version.

Theorem 2. *The running time of the algorithm is $O(N \log N)$, where $N = \sum_{i=1}^n c_i \times m \times (r + 1)$.*

Notice that although in the JIS problem, a multi-machine scheduling problem can be converted into a single-machine scheduling problem without affecting the quality of solutions [3], in JISMC problem such a conversion in general does not work as it will violate the consistency condition.

4 Scheduling Algorithm for Multi-type of Resources with Dependency Relation

In this section, we consider the scheduling problem for multiple types of resources among which a dependency relation exists. For simplicity, we assume that all machines are identical and there is only one copy in each type of resource (such assumptions can be easily removed without affecting the quality of solution).

Let $\mathcal{R} = \{R_1, R_2, \dots, R_r\}$ be the set of r resources. Recall that a resource R_j depends on another resource R_l if for each job $J_i \in \mathcal{J}$, $R_j \in \mathcal{R}_i$ implies $R_l \in \mathcal{R}_i$ and every reservation intervals of J_i on R_j and R_l are the same. Based on the dependency relation, a dependency graph $G_{\mathcal{R}}$ can be formed by connecting a directed edge from each $R_j \in \mathcal{R}$ to every resource which R_j directly depends on. Without loss of generality, we can assume that $G_{\mathcal{R}}$ is a directed acyclic graph (DAG). Otherwise, if there is a cycle C in $G_{\mathcal{R}}$, then by the definition of dependency relation, for each job J_i either all resources on C will be in \mathcal{R}_i or none of them will be in \mathcal{R}_i . Thus all resources on C can be collapsed as a single resource.

A resource R_j is called the *anchor resource* of a job J_i if R_j depends on every other resource directly or indirectly in \mathcal{R}_i . In general, a job may not have an anchor resource. In this paper, we only consider the case in which each job has exactly one anchor resource.

To solve this version of scheduling problem, our main idea is to generalize our algorithm for the case of single resource. For each job J_i , we first generate a set of machine intervals \mathcal{I}_i^M and a set of resource intervals \mathcal{I}_i^R based on the requested resources \mathcal{R}_i .

In the evaluation phase, we consider the linear order of all intervals based on their ending times. This linear order captures all possible contentions among intervals while competing for machines. Similar to the algorithm for single resource,

the evaluation phase first initializes a stack S_{M_j} for each machine $M_j, 1 \leq j \leq m$, and a stack S_{R_k} for each resource R_k for $1 \leq k \leq r$. Then it processes all intervals according to this linear order. For each interval $I \in J_i$, the algorithm identifies a set $C(I)$ of all conflicting intervals from all stacks. $C(I)$ includes three types of intervals: (a) Intervals from the same family (i.e., the same job); (b) Intervals overlapping with I on the same machine; (c) Intervals conflicting with I on any resource required by I . Based on all intervals in $C(I)$, the algorithm computes a potential value $I.value(j)$ for I on each machine M_j by subtracting the values of all conflicting intervals in $C(I)$ from the profit of I (i.e., $I.value(j) = w_i + \Delta_i^w - \sum_{I' \in C(I)} I'.value$), and sets the value $I.value$ to be the maximum potential value among the m machines. If $I.value$ is positive, then I is pushed into the stack of the machine which has the maximum potential value for I , and into stacks of the resources in \mathcal{R}_i . The evaluation procedure repeats this procedure until all intervals are finished.

The selection phase is similar to the one for the case of single resource. The only difference is that when an interval I is selected, it blocks all intervals in the stacks which conflict with I on the family of I , on the machine executing I , and on all resources requested by I .

To estimate the quality of the obtained scheduling, we first partition the conflicting set $C(I)$ of each interval $I \in J_i$ into 3 subsets, $C_f(I), C_{M_j}(I)$ and $C_R(I)$, where $C_f(I)$ includes all intervals in $C(I)$ which are from the same family of J_i , $C_{M_j}(I)$ is the set of intervals in $C(I) \setminus C_f(I)$ which are conflicting with I on machine M_j , and $C_R(I)$ is the set of intervals in $C(I) \setminus (C_f(I) \cup C_{M_j}(I))$ which are conflicting with I on some resources. The following lemma is a key to prove the quality of our scheduling.

Lemma 5. $I_1 \in J_{i_1}, I_2 \in J_{i_2}, A = C_f(I_1) \cup C_f(I_2) \cup C_{M_{j_1}}(I_1) \cup C_{M_{j_2}}(I_2) \cup C_R(I_1) \cup C_R(I_2)$

The proof for $C_f(I_1) \cap C_f(I_2) = \phi$ and $C_{M_{j_1}}(I_2) \cap C_{M_{j_2}}(I_2) = \phi$ is the same as in Lemma 2. To prove $C_R(I_1) \cap C_R(I_2) = \phi$, we consider two cases: (1) $\mathcal{R}_{i_1} \cap \mathcal{R}_{i_2} = \phi$ and (2) $\mathcal{R}_{i_1} \cap \mathcal{R}_{i_2} \neq \phi$. For the first case, clearly we have $C_R(I_1) \cap C_R(I_2) = \phi$. For the second case, let R_j be any resource in $\mathcal{R}_{i_1} \cap \mathcal{R}_{i_2}$. Then, since I_1 and I_2 are two intervals from a feasible scheduling, the reservation intervals of I_1 and I_2 on R_j must be disjoint. By the consistency property, reservation intervals on R_j will have the same order as that of the corresponding resource intervals on any machine. Thus intervals conflicting with I_1 on R_j will not conflict with I_2 on R_j . Thus $C_R(I_1) \cap C_R(I_2) = \phi$.

Similar to the lemma 1 and 3, we have the following lemmas and theorems.

Lemma 6. $\mathcal{R} = \mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \mathcal{R}_{i_3} \cup \dots \cup \mathcal{R}_{i_m}$

Lemma 7. $\mathcal{R} = \mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \mathcal{R}_{i_3} \cup \dots \cup \mathcal{R}_{i_m} \implies V(S) + V(C_F) + V(C_R) \geq P(A)$

Theorem 3.

Theorem 4. $O(N \log N)$
 $N = \sum_{i=1}^n c_i \times m \times (|\mathcal{R}_i| + 1)$

5 Scheduling Algorithm for Multi-types of Resources with No Dependency Relation

In this section, we consider the scheduling problem of multiple types of resources with no dependency relation. We assume the same settings as in last section except for the dependency relation. The algorithm for solving this version of problem is the same as the one given in last section. The main difference is on the analysis.

To estimate the quality of the scheduling, the key is to be able to show that the conflicting set $C(I)$ can be partitioned into classes such that the total value of all intervals in each class can be bounded. For this purpose, we partition all intervals in $C(I)$ as follows.

1. $C_f(I)$: the set of intervals generated from the same job as I and are pushed into any of the m stacks of machines before evaluating I .
2. $C_{M_j}(I)$: the set of intervals inserted into S_{M_j} before evaluating I and overlapping with I on time.
3. $C_{R_1}(I)$: the set of intervals which compete for resource R_1 with I , if R_1 is not requested by I , then it is an empty set.
4. $C_{R_2}(I)$: the set defined similarly as $C_{R_1}(I)$ but competing for resource R_2 .
5. :
6. $C_{R_r}(I)$: the set defined similarly as $C_{R_1}(I)$ but competing for R_r .

Notice that in the above definition, an interval in $C(I)$ could belong to more than one subsets if it has multiple contentions with I . To avoid this problem, we give an order to them as $C_f(I), C_{M_m}(I), C_{M_{m-1}}(I), \dots, C_{M_1}(I), C_{R_1}(I), \dots, C_{R_r}(I)$. To partition $C(I)$, we first pick intervals for $C_f(I)$ from $C(I)$, then for $C_{M_m}(I)$ from the set $C(I) \setminus C_f(I)$, and so on in this order. In this way, no interval in $C(I)$ will appear in more than one subset.

With the above partition, we have the following lemma.

Lemma 8. $I_1 \dots I_2 \dots A$
 $C_{M_{j_1}}(I_1) \dots C_{M_{j_2}}(I_2), C_{R_k}(I_1) \dots C_{R_k}(I_2) \dots M_{j_1}$
 $M_{j_2} \dots I_1 \dots I_2 \dots A, \dots$
 $R_k \dots$

With the above lemma, we are able to prove the following theorem.

Theorem 5. $2+r$

Although the above ratio holds for the most general case, it may not reflect the actual quality of some schedulings. To obtain a better approximation ratio, we consider the following undirected graph $G_{\mathcal{R}}^C$, called *potential conflicting graph*. For each job J_i , connect all resources in \mathcal{R}_i into a tree in an arbitrary order. The union of the n trees forms the potential conflicting graph. The following lemma gives some nice properties of such graphs.

Lemma 9. Let C_1, C_2, \dots, C_r be maximal connected components of $G_{\mathcal{R}}^C$. For each $j \in \{1, 2, \dots, r\}$, let $R_{j_1}, R_{j_2}, \dots, R_{j_{|C_j|}}$ be the resources in C_j . Let $I_1 \in J_{j_1}, I_2 \in J_{j_2}, \dots, I_{|C_j|} \in J_{j_{|C_j|}}$ be intervals that use resources $R_{j_1}, R_{j_2}, \dots, R_{j_{|C_j|}}$ respectively. Then $C_{R_{j_1}}(I_1) \cap C_{R_{j_2}}(I_2) = \emptyset$.

First we can assume that R_{j_1} is requested by I_1 since otherwise $C_{R_{j_1}}(I_1)$ would be an empty set and the lemma will be trivially true. Similarly, we assume that R_{j_2} is requested by I_2 . Let I_3 be any interval which competes with I_1 for resource R_{j_1} . Then all the resources I_3 requested will come from the same connected component C_1 . Since if otherwise it uses a resource outside C_1 , C_1 will not be a maximal connected component of $G_{\mathcal{R}}^C$. By the same argument, any interval that competes with I_2 for resource R_{j_2} can only use resources in C_2 , which means that I_3 does not belong to $C_{R_{j_2}}(I_2)$. The other direction can be proved similarly. Hence the two sets are disjoint.

With the above lemma, we have the following packing lemma.

Lemma 10. Let A be a set of intervals and C be a set of resources. Let $C_{R_j} = \bigcup_{I \in A} C_{R_j}(I)$, $1 \leq j \leq r$. Let $S = \bigcup_{j=1}^r C_{R_j}$. Then $|S| \leq m \cdot |C|$, where $m = \max_{1 \leq j \leq r} |C_j|$.

Let R_1, R_2, \dots, R_k be the k resources in C . For any interval I in A , it can only use resources from a single connected component. If $C_{R_i}(I) \neq \emptyset$ and $R_i \in C_i$, then for any resource $R_j \notin C_i$, $C_{R_j}(I) = \emptyset$. The k sets of intervals can be constructed as follows. Initially, let $C_{R_i} = \bigcup_{I \in \mathcal{I}} C_{R_i}(I)$, $1 \leq i \leq k$, where \mathcal{I} is the set of intervals in A which use at least one resource in C . Note that \mathcal{I} will not use any resource outside R_i , $1 \leq i \leq k$.

Let C_2 be another connected component of size l ($l \leq k$) and $R_p, R_{p+1}, \dots, R_{p+l-1}$ be its resources. By Lemma 9, for any interval I using resource R_p , $C_{R_1} \cap C_{R_p}(I) = \emptyset$. We can merge $C_{R_p}(I)$ into C_{R_1} , i.e., $C_{R_1} = C_{R_1} \cup C_{R_p}(I)$. Similarly, we can also merge $C_{R_{p+1}}(I)$ into C_{R_2} , $C_{R_{p+2}}(I)$ into C_{R_3} , \dots , $C_{R_{p+l-1}}(I)$ into C_{R_l} . Since $l \leq k$, each set $C_{R_{p+i}}$ can be merged into one of the k sets $C_{R_{i+1}}$, $1 \leq i \leq l-1$.

Repeating the same procedure for all other components, we can eventually pack each interval in S into one of the k sets. Clearly, $C_{R_i} \subseteq S$, $1 \leq i \leq k$.

With the packing lemma, we can reduce approximation ratio from $r + 2$ to $k + 2$.

Theorem 6.

$$G_{\mathcal{R}}^C \leq \frac{k+2}{k} G_{\mathcal{R}}^C$$

6 Application in Optical Burst Switching Networks

In this section, we show that our JISMC model and its techniques can be easily applied to Optical Burst Switching (OBS) networks for developing more efficient algorithms for an important channel scheduling problem.

As a promising paradigm for the next-generation Internet, OBS [11, 15] has attracted a great deal of attentions in recent years. A major problem in such networks is to efficiently assign channels (wavelength) to incoming bursts (packets) so that the total bandwidth utility is maximized and consequently the burst loss rate is minimized. In an OBS network, an ingress OBS node assembles data into data bursts, and sends out a control packet for each burst. The control packet precedes the corresponding data burst by an offset time. Based on the information contained in the control packet, a reservation for a period of time equal to the burst length (starting at the expected burst arrival time) can be made at each node. Due to high burst loss rate in OBS, Fiber Delay Lines (FDL) are often used to delay bursts by a fixed amount of time. If an FDL of d unit(s) of delay is assigned to a burst, then the scheduler can reserve the time slot which is d unit time later.

A number of channel scheduling algorithms [12, 13, 14] have been proposed to tackle this problem. Among them Min-SV algorithm proposed by Xu [14] has so far the best performance *w.r.t* both the scheduling time and loss rate. One common feature of the above algorithms is that they all use certain greedy strategies to schedule bursts in a purely on-line fashion. That is, they schedule each burst immediately upon receiving its corresponding control packet without using any information about future bursts. Due to this reason, a “wrongly” scheduled burst could block a number of future incoming bursts, making the overall quality of scheduling rather poor. This is especially the case when links are heavily loaded. Hence, certain global optimization based scheduling is desired.

In OBS networks, we can model channels as identical machines, bursts as jobs of fixed starting and ending time, and the FDLs as resources. Clearly, the resources are consistent with the machines because each burst use an FDL immediately before using an outgoing channel. The only problem is that bursts come in an online fashion, the scheduling is based only on past information, and nothing is known about future bursts. Our approximation algorithms, however, assume that all the jobs are known in advance. To overcome this difficulty, the main idea is to collect bursts “periodically”, form a batch of bursts based on their timing, and schedule each batch of bursts as a whole by our algorithm in Section 5.

Experiments show that our batching-based algorithm significantly outperforms Min-SV [14] by a factor of almost 20% and consistently outperforms the other batching-based algorithm. (More experimental results and the details of our scheduling algorithm are left for the full paper.)

References

1. Karhan Akcoglu, James Aspnes, Bhaskar DasGupta, and Ming-Yang Kao. Opportunity cost algorithms for combinatorial auctions. In *Applied Optimization: Computational Methods in Decision-Making, Economics and Finance*, pages 455–479. Kluwer Academic Publishers, 2002.
2. Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 735–744. ACM Press, 2000.
3. Piotr Berman and Bhaskar DasGupta. Improvements in throughput maximization for real-time scheduling. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 680–687. ACM Press, 2000.
4. Piotr Berman, Bhaskar DasGupta, and S. Muthukrishnan. Simple approximation algorithm for nonoverlapping local alignments. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 677–678, 2002.
5. Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. In *IEEE Symposium on Foundations of Computer Science*, pages 348–356, 2001.
6. B. DasGupta and M. A Palis. Online real-time preemptive scheduling of jobs with deadlines on multiple machines. *Journal of Scheduling*, 4(6):297–312, 2001.
7. Thomas Erlebach and Frits C.R. Spieksma. Simple algorithms for a weighted interval selection problem. *Eleventh Annual International Symposium on Algorithms And Computation (ISAAC 2000)*, pages 228–240, 2000.
8. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of the ACM*, 34:144–162, 1987.
9. Spyros Kontogiannis. Lower bounds & competitive algorithms for online scheduling of unit-size tasks to related machines. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 124–133. ACM Press, 2002.
10. Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 302–311. ACM Press, 1994.
11. Chunming Qiao and Myungsik Yoo. Optical burst switching(obs) - a new paradigm for an optical internet. In *Journal High Speed Networks*, volume 8, pages 69–84, 1999.
12. J. Turner. Terabit burst switching. *Journal High Speed Networks*, 8:3–16, 1999.
13. Yijun Xiong, Marc Vandenhouste, and Hakki C. Cankaya. Control architecture in optical burst-switched wdm networks. In *IEEE Journal On Selected Areas in Communications*, volume 18, pages 1838–1851, Oct 2000.
14. Jinhui Xu, Chunming Qiao, Jikai Li, and Guang Xu. Efficient channel scheduling algorithms in optical burst switched networks. In *INFOCOM 2003, 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, San Francisco, March 2003.
15. M. Yoo and C. Qiao. A high speed protocol for bursty traffic in optical networks. *SPIE's All-Optical Communication Systems:Architecture, Control and Protocol Issues*, 3230:79–90, Nov, 1997.

Superimposing Voronoi Complexes for Shape Deformation

Chao Chen and Ho-Lun Cheng

National University of Singapore, Singapore

Abstract. Edelsbrunner et al. defined a framework of shape deformations with shapes bounded by skin manifold. We prove that the infinitely many synthesized shapes in the deformation sequence share finitely many common Voronoi complexes. Therefore, we propose a new algorithm to compute the common Voronoi complexes efficiently for the deformations, and use these common complexes to compute the synthesized shapes in real time. This makes generating, visualizing, and customizing shape deformations feasible.

1 Introduction

Edelsbrunner et al. defined a framework of shape space construction [2]. See Figure 1 for an example. Given two reference shapes which are represented by the skin manifold, a smooth deformation can be computed automatically and robustly. This shape space construction provides a new paradigm for geometric modeling, animation designs and molecular motion simulation. However, it is limited to only two shapes, and also, not feasible for real time visualization.

In this work, we extend this blending of shapes to n reference shapes in any dimension. Imagine that given three faces represented by skin surfaces, we can synthesize any new face with features of the reference faces. Moreover, because of the extension to n shapes, we are able to customize the shape deformations by introducing additional reference shapes. In the example of Figure 2, we influence the deformation of the shape \mathbf{X} to \mathbf{I} with a shape \mathbf{O} . By applying a positive influence of the shape \mathbf{O} , we can create a hole during the deformation. On the contrary, if we apply a negative influence of the shape \mathbf{O} , we can compact the deforming shape.

We also make the visualization feasible for real time applications. The bottleneck lies on the generation of the Voronoi complex of each instance of the deforming shapes. The reason is that the shapes are represented by the skin surfaces and they are built on the Delaunay complexes of weighted point sets. It means that for each instance in the morphing, a Voronoi complex (or its Delaunay triangulation) is required. See Figure 1 for an example. The deformation of a shape \mathbf{X} into another shape \mathbf{I} requires a sequence of intermediate shapes $(1-t)\mathbf{X} + t\mathbf{I}$, for $t \in [0, 1]$. For each instance in the deformation, it takes $O(m \lg m + m^{\lceil d/2 \rceil})$ time to construct its Voronoi complex in \mathbb{R}^d , in which m is

the cardinality of the weighted point set. Moreover, we usually need a lot of instances in the sequence of deformation to generate a smooth visualization effect. Thus, it is infeasible to compute the intermediate shapes in real time.

Contribution. In this paper, we improve the efficiency of the intermediate shapes computation and extend the works of Edelsbrunner:

1. For each Voronoi complex construction, we improve the bound to $O(m^3)$ in any dimension, if the Voronoi complexes of the initial shapes are given.
2. We prove that all the infinite number of intermediate shapes share finitely many Voronoi complexes.
3. We generalize Edelsbrunner’s framework to more than two shapes, together with the additional combinatorial structures required in the new type of degenerated Voronoi complexes.

The first two points make it feasible for visualizing the deformation in real time. For example, the Voronoi complexes for the shapes in Figure 1 are identical for $0 < t < 1$. Thus, we generate only one Voronoi complex for all the shapes in the deformation sequence.

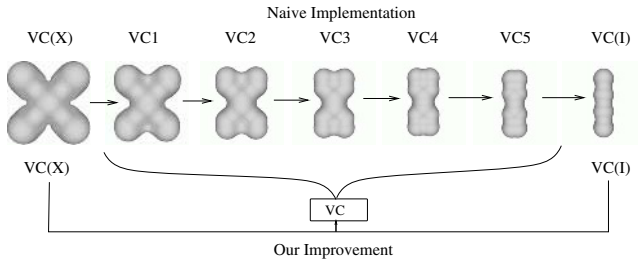


Fig. 1. A deformation of the shape \mathbf{X} into \mathbf{I} in \mathbb{R}^3

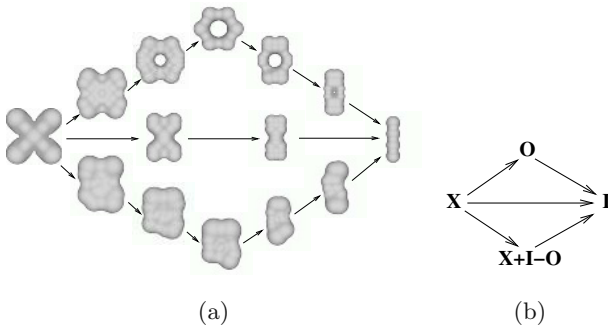


Fig. 2. Three deformations of \mathbf{X} into \mathbf{I} influenced by \mathbf{O} in three different ways

Outline. In Section 2, we introduce the skin and its construction. In Section 3, we define the the intermediate shape, and present a theorem stating that there are only finitely many Voronoi complexes for all the intermediate shapes. Finally, in Section 4, we design a new algorithm to compute the Voronoi complexes of the intermediate shapes.

2 Backgrounds

We introduce the geometric foundation of the skin body that is bounded by the ... , a compact manifold without boundary in any dimension. It is defined by a weighted point set, its Voronoi and Delaunay complexes. This section serves for the purpose of stating the notations for later sections. Readers who want to know more details about the skin surface should refer to [4]. At the end of this section, we will also introduce the notation of furthest-neighbor Voronoi complex, which is useful in later sections when we extend the morphing with n reference shapes.

Delaunay and Voronoi Complexes. The skin surface is built on the structures of Delaunay and Voronoi complexes of a weighted point set. We first define some notations for the Delaunay and Voronoi complexes.

A \bullet ... in \mathbb{R}^d can be written as $b_i = (z_i, w_i) \in \mathbb{R}^d \times \mathbb{R}$, where $z_i \in \mathbb{R}^d$ is its position and $w_i \in \mathbb{R}$ is its weight. We can also view a weighted point, b_i , as a ball in \mathbb{R}^d with center z_i and radius $\sqrt{w_i}$. For any $B_0 \subseteq \mathbb{R}^d \times \mathbb{R}$, we denote $z(B_0)$ as the set of the centers of the weighted points in B_0 . The \bullet ... of a point $x \in \mathbb{R}^d$ from a weighted point, b_i , is defined as

$$\pi_{b_i}(x) = \|xz_i\|^2 - w_i. \tag{1}$$

Given a finite set B_0 of n weight points, the \bullet ... , ν_i , for each weighted point, $b_i \in B_0$, is

$$\nu_i = \{x \in \mathbb{R}^d \mid \pi_{b_i}(x) \leq \pi_{b_j}(x), b_j \in B_0\}.$$

We define the non-empty intersection of m Voronoi regions as the \bullet ... of a set of weighted points $X \subseteq B_0$, namely, $\nu_X = \bigcap_{b_i \in X} \nu_i$. The collection of all the non-empty Voronoi cells is called the \bullet ... of B_0 , denoted as V_{B_0} . For each $\nu_X \in V_{B_0}$, its corresponding \bullet ... , δ_X , is the convex hull of the set of centers of X , namely, $\text{conv}(z(X))$. The collection of all the Delaunay cells is called the \bullet ... of B_0 , denoted as D_{B_0} .

Given a Delaunay cell δ_X , its \bullet ... , $\dim(\delta_X)$, is that of the affine hull of δ_X . If $\dim(\delta_X)$ is no more than $\text{card}(X) - 1$, δ_X is a simplex. If this is true for all cells in D_{B_0} , it is a \bullet The Delaunay complex is simplicial under the following general position assumption in \mathbb{R}^d .

ASSUMPTION 1 (GENERAL POSITIONS) $\forall \nu_X \in V_{B_0}, \text{card}(X) = \dim(\delta_X) + 1$.

The dimension of ν_X , $\dim(\nu_X)$, is $d - \dim(\delta_X)$. We call ν_X a \bullet ... if its corresponding Delaunay cell is a simplex.

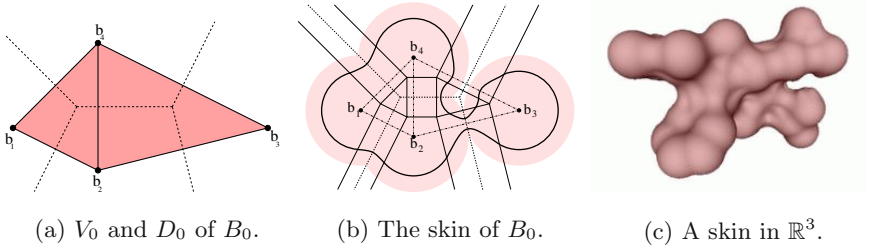


Fig. 3. The left subfigure is the Voronoi and Delaunay complexes of a weighted point set in \mathbb{R}^2 , $B_0 = \{b_1, b_2, b_3, b_4\}$. The center subfigure is the skin of B_0 , which is constructed on the complexes of B_0 . The right subfigure is the skin body of a molecule in \mathbb{R}^3

We can define a new Voronoi complex using a different distance function. If we employ $-\pi_{b_i}(x)$ as the distance function, the resulting Voronoi complex is the furthest-neighbor Voronoi complex. Each of its elements contains points with the longest weighted distances from the weighted points. The furthest-neighbor Voronoi complex will be used to construct the Voronoi complex of the intermediate shape in the next section.

Skin. A skin surface is Specified by a set of weighted points $B_0 = \{b_i \in \mathbb{R}^d \times \mathbb{R} \mid i = 1..n_0\}$. Before defining the skin, three operations on weighted points (or balls) are given. The aim is to establish the operations for linear combinations of weighted points [4] as in points in \mathbb{R}^d . Also, these operations are useful in the proof in later sections. For $b_i, b_j \in B_0$ and $\gamma \in \mathbb{R}$, the addition, scalar multiplication, and square root of weighted points are defined as

$$\begin{aligned}
 b_i + b_j &= (z_i + z_j, w_i + w_j + 2\langle z_i, z_j \rangle), \\
 \gamma b_i &= (\gamma z_i, \gamma w_i + (\gamma^2 - \gamma)\|z_i\|^2), \quad \text{and} \\
 \sqrt{b_i} &= (z_i, w_i/2),
 \end{aligned}$$

respectively, in which $\langle z_i, z_j \rangle$ is the scalar product of two vectors z_i and z_j . A convex combination of B_0 is $\sum_{i=1}^{n_0} \gamma_i b_i$ with $\gamma_i \in \mathbb{R}$ and $i = 1..n_0$. It is a convex combination if $\sum_{i=1}^n \gamma_i = 1$ and all $\gamma_i \geq 0$. Thus, we can define the convex hull of B_0 as $\text{conv}(B_0)$, namely, the set of all the convex combinations of B_0 .

Now, we are ready to define the skin surface. We first shrink all the balls in $\text{conv}(B_0)$ by the square root operations. Then, the skin surface is the envelop of the underlying space of these shrunken balls, formally,

$$\text{skin}(B_0) = \partial \left(\bigcup \sqrt{\text{conv}(B_0)} \right).$$

For the space bounded by the skin surface, we call this the skin body, denoted as $\text{body}(B_0) = \bigcup \sqrt{\text{conv}(B_0)}$. Figure 3(c) shows an example of the skin body in \mathbb{R}^3 .

Skin Decomposition. Here, we discuss the skin decomposition and give the reason why we need the Delaunay and Voronoi complexes for the skin construction. The skin of B_0 can be decomposed by \dots . A mixed cell, μ_X , is a Minkowski sum of a Delaunay cell and its corresponding Voronoi cell, namely, $\mu_X = (\nu_X + \delta_X)/2$. All the mixed cells partition \mathbb{R}^d and decompose the skin surface. Within each mixed cell, $\text{skin}(B_0) \cap \mu_X$ is a quadratic surface in the form of $\sum_{i=1}^d \pm x_i^2 = w_X/2$ after translation to the \dots of μ_X . The center and size of μ_X is defined as

$$z_X = \text{aff}(\delta_X) \cap \text{aff}(\nu_X), \quad \text{and} \tag{2}$$

$$w_X = w_i - \|z_X z_i\|^2, \tag{3}$$

respectively. In \mathbb{R}^2 , there are three types of mixed cell and the ‘patch’ is either a portion of a circle or a hyperbola clipped with a mixed cell. See Figure 3(b) for a decomposed skin.

3 Intermediate Shapes and Their Voronoi Complexes

In deformation, we start with some reference shapes and generate some mixtures of them, namely, \dots . After discussing how they are generated, we present the theorem of all the intermediate shapes share finitely many Voronoi complexes.

Intermediate Shapes. Given a finite set of given shapes, namely, the \dots , an infinite family of intermediate shapes can be constructed. Each intermediate shape is the mixture of the reference shapes, with a set of weights signaling the similarities between the mixture and the reference shapes.

Let the number of reference shapes be n . Let $\mathfrak{B} = \{B_1, \dots, B_n\}$ be the set of weighted point sets defining the set of reference shapes, $S = \{\text{body}(B_i) \mid B_i \in \mathfrak{B}\}$. To define the intermediate shapes of S , we firstly define the affine combinations of \mathfrak{B} , namely, $B(c)$.

Given two weighted point sets, B_0 and B_1 , we define the \dots , and \dots of them as

$$B_0 + B_1 = \{b_i + b_j \mid b_i \in B_0, b_j \in B_1\}, \quad \text{and}$$

$$\gamma B_0 = \{\gamma b_i \mid b_i \in B_0\}, \text{ for } \gamma \in \mathbb{R},$$

respectively. With these operations, we can define the \dots of \mathfrak{B} as $B(c) = \sum_{i=1}^n \gamma_i B_i$ with the \dots , $c = (\gamma_1, \dots, \gamma_n) \in \mathbb{R}^n$. If $\sum_{i=1}^n \gamma_i = 1$, $B(c)$ is an \dots of \mathfrak{B} , namely, the \dots . Note that the cardinality of $B(c)$ is no more than $\prod_{i=1}^n \text{card}(B_i)$. We do not limit the weights to be positive for a convex combination because negative weights play an interesting role in shape mixing.

An \dots is defined as the skin shape bounded by the skin of $B(c)$, namely, $\text{body}(B(c))$. Each intermediate shape corresponds to a coefficient

vector, c , which states the weights of all these reference shapes in this mixture. All the coefficient vectors of the intermediate shapes form a hyperplane $\Gamma \subseteq \mathbb{R}^n$. A transition of one intermediate shape into another is parameterized with a path connecting the two corresponding coefficient vectors in Γ . Note that each reference shape is also an intermediate shape, whose corresponding coefficient vector is a unit basis vector in \mathbb{R}^n .

Intermediate Voronoi Complexes. We denote the Voronoi complex of $B(c)$ as $V(c)$, namely, the set of all Voronoi regions. To compute an intermediate shape, $\text{body}(B(c))$, the corresponding intermediate Voronoi complex, $V(c)$, is required. However, it is computationally expensive if we compute $V(c)$ for each individual value of c , especially in deformation of the intermediate shapes. On the other hand, we will show that Γ can be divided into finite number of partitions, and in each partition, the intermediate Voronoi complexes are the same for all the possible values of c .

Each weighted point $b(c) \in B(c)$ is an affine combination of n weighted points from the n given $B_i \in \mathfrak{B}$, namely, $b(c) = \sum_{i=1}^n \gamma_i b_i$, for some $b_i \in B_i$. In order to define the Voronoi region of $b(c)$ with respect to $B(c)$, we firstly derive the weighted distance of any point $x \in \mathbb{R}^d$ from $b(c)$.

Lemma 1. For $b(c) = \sum_{i=1}^n \gamma_i b_i$, $\sum_{i=1}^n \gamma_i = 1$, and $x \in \mathbb{R}^d$, $b(c) = \sum_{i=1}^n \gamma_i \pi_{b_i}(x)$, where $\pi_{b_i}(x)$ is the distance of x from b_i, \dots, b_n .

$$\pi_{b(c)}(x) = \sum_{i=1}^n \gamma_i \pi_{b_i}(x).$$

We prove this lemma by induction on the number n . For $n = 1$, it is true since $\gamma_1 = 1$ and $B(c) = B_1$. When $n = 2$, we have

$$\pi_{b(c)}(x) = \gamma_1 \pi_{b_1}(x) + (1 - \gamma_1) \pi_{b_2}(x), \tag{4}$$

by using Equation (1). The claim is true for $n = 2$.

Assume the claim is true for $n = k$. When $n = k + 1$, without loss of generality, we assume $\gamma_{k+1} \neq 1$. Let $\mathfrak{B}' = \mathfrak{B} - \{B_{k+1}\}$ and

$$c' = \left(\frac{\gamma_1}{1 - \gamma_{k+1}}, \frac{\gamma_2}{1 - \gamma_{k+1}}, \dots, \frac{\gamma_k}{1 - \gamma_{k+1}} \right).$$

Let $B'(c')$ be the affine combination of \mathfrak{B}' , and $b'(c') \in B'(c')$. Then, $b(c)$ can be expressed as the affine combination of $b'(c')$ and b_{k+1} , namely,

$$b(c) = (1 - \gamma_{k+1})b'(c') + \gamma_{k+1}b_{k+1}.$$

Follow Equation (4), the weighted distance of any point $x \in \mathbb{R}^d$ from $b(c)$ is

$$\pi_{b(c)}(x) = (1 - \gamma_{k+1})\pi_{b'(c')}(x) + \gamma_{k+1}\pi_{b_{k+1}}(x) = \sum_{i=1}^{k+1} \gamma_i \pi_{b_i}(x),$$

as required. □

We are now ready to give a new method to determine the Voronoi region of $b(c)$ with respect to $B(c)$. We start constructing $V(c)$ from a simple situation, in which $\gamma_i > 0$ for all i .

Lemma 2. Let $b(c) \in B(c)$, $\gamma_i > 0$, $B(c) = \{b_1, \dots, b_n\}$, B_1, \dots, B_n .

$$\nu_{b(c)} = \bigcap_{i=1}^n \nu_{b_i}. \tag{5}$$

It is easy to see that $\bigcap_{i=1}^n \nu_{b_i} \subseteq \nu_{b(c)}$, because $\gamma_i \pi_{b_i}(x) \leq \gamma_i \pi_{b'_i}(x)$ for all $b'_i \in B_i$. Next, we prove $\nu_{b(c)} \subseteq \bigcap_{i=1}^n \nu_{b_i}$. We will show that for $k = 1..n$, and any point $x \in \mathbb{R}^d$, $x \in \nu_{b(c)}$ implies that $x \in \nu_{b_k}$. For any $b'_k \in B_k$, let

$$b'(c) = \sum_{i=1..n, i \neq k} \gamma_i b_i + \gamma_k b'_k.$$

We have $\pi_{b(c)}(x) \leq \pi_{b'(c)}(x)$ if $x \in \nu_{b(c)}$, that is,

$$\sum_{i=1}^n \gamma_i \pi_{b_i}(x) \leq \sum_{i=1..n, i \neq k} \gamma_i \pi_{b_i}(x) + \gamma_k \pi_{b'_k}(x).$$

Simplifying this inequality, we have $\gamma_k \pi_{b_k}(x) \leq \gamma_k \pi_{b'_k}(x)$, which implies $x \in \nu_{b_k}$, as required. □

Next, we generalize this lemma to any possible values of c in Γ . We assume $\gamma_i \neq 0$ for any i , because for any $\gamma_k = 0$,

$$B(c) = \sum_{i=1..n, i \neq k} B_i.$$

If $\gamma_i < 0$, the lemma is true if we substitute γ_k with $-\gamma_k$, and $\pi_{b_k}(x)$ with $-\pi_{b_k}(x)$. That means, the lemma remains true if we use $-\pi_{b_k}(x)$ as the distance function, and the furthest-neighbor Voronoi region as ν_{b_k} .

For convenience, we assign signs to the Voronoi complex and its elements. For a weighted point set, B_0 , its Voronoi complex with ‘+’ sign, $V_{B_0}^+$, is the nearest-neighbor Voronoi complex of B_0 , and $V_{B_0}^-$ is the furthest-neighbor Voronoi complex of B_0 . The same rule applies to the Voronoi cells ν_X^+ and ν_X^- .

Therefore, the Voronoi cell of $b(c)$ with respect to $B(c)$ is the intersection of the signed Voronoi cells of b_i with respect to B_i , whose signs are determined by the signs of the corresponding γ_i , namely,

$$\nu_{b(c)} = \bigcap_{i=1}^n \nu_{b_i}^{\text{Sign}(\gamma_i)}.$$

This leads to the following Theorem about the intermediate Voronoi complex.

Theorem 1.

$$V(c) = \{ \nu_{X(c)} = \bigcap_{i=1}^n \nu_{X_i}^{Sign(\gamma_i)} \mid \nu_{X(c)} \neq \emptyset, X(c) = \sum_{i=1}^n \gamma_i X_i, X_i \subseteq B_i \}.$$

According to this theorem, an intermediate Voronoi cell is the collection of the non-empty intersections of n signed Voronoi cells, from the n signed Voronoi complexes of the reference shapes, respectively. See Figure 4 for an example of superimposing two signed Voronoi complexes when all the coefficients are positive.

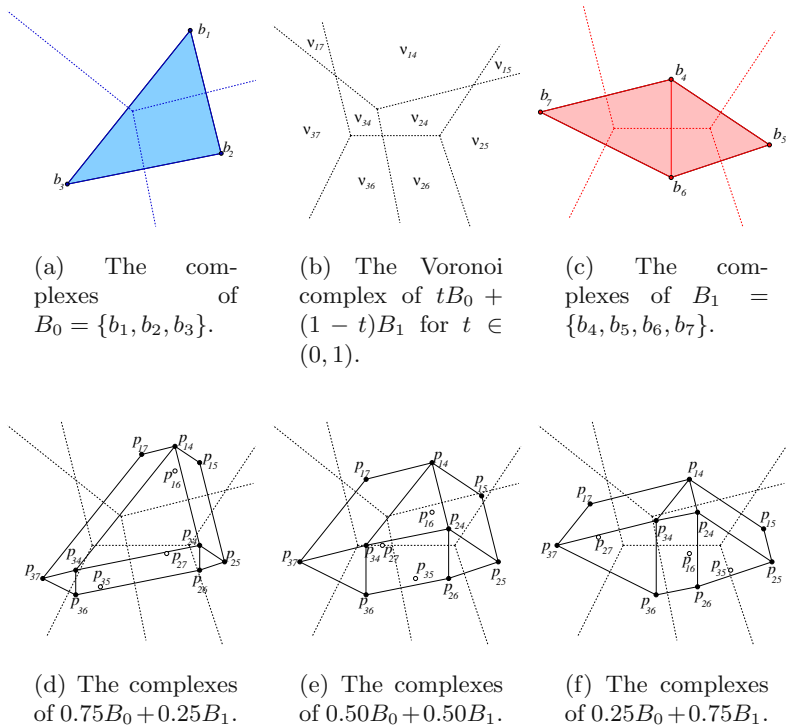


Fig. 4. Deformation of the Delaunay complexes. The Voronoi complex remains the same and it the superimposition of two signed Voronoi complexes

The intermediate Voronoi complexes, $V(c_0)$ and $V(c_1)$, are the same if $Sign(\gamma_{0,i}) = Sign(\gamma_{1,i})$ for all i , in which $\gamma_{0,i}$, $\gamma_{1,i}$ are the i -th coordinates of c_0 and c_1 respectively. By this, we can divide Γ into $2^n - 1$ convex partitions with respect to the signs of the coefficients. The number of partitions is $2^n - 1$ rather than 2^n , because the coordinates of a coefficient vector $c \in \Gamma$ can not be all negative.

Each partition covers all the values of coefficient vectors, $c = (\gamma_1, \dots, \gamma_n)$, with the same $(\text{Sign}(\gamma_1), \dots, \text{Sign}(\gamma_n))$. For any two coefficient vectors in the same partition, their corresponding intermediate Voronoi complexes are the same.

For a deformation of one intermediate shape into another, its parameterizing path is also divided into finitely many pieces. For all the coefficient vectors lie in a same piece, their corresponding intermediate shapes share a same intermediate Voronoi complex. See Figure 4.

Enumerating Voronoi Cells in $V(c)$. In \mathbb{R}^d , a Voronoi cell, ν_X , is the non-empty intersection of m Voronoi regions. If $m > d + 1 - \dim(\nu_X)$, we call ν_X degenerate. Otherwise, it is simplicial as we defined in the background section. In the intermediate Voronoi complex, the degenerate case is unavoidable. See Figure 4 for an example in \mathbb{R}^2 . When $n = 2$ and $d = 2$, the intersection of two Voronoi edges is a Voronoi vertex, which is the common intersection of the four Voronoi regions of the four weighted points in the intermediate weighted point set. It is degenerate because it is the intersection of four Voronoi regions while its dimension, $\dim(\nu_{X(c)}) = 0$. There are three such degenerate Voronoi cells in the intermediate Voronoi complex in Figure 4.

We classify all the Voronoi cells in $V(c)$, each of which is either simplicial or degenerate, with regard to the dimensions of their contributors. Let τ_j be the number of contributors whose dimensions are j . We define the type of a Voronoi cell as a tuple, $(\tau_0, \dots, \tau_{d-1})$. The variable τ_d is omitted, because it is dependent, namely, $\tau_d = n - \sum_{j=0}^{d-1} \tau_j$. Enumerating all the types of $\nu_{X(c)}$ is equivalent to enumerating all the tuples, $(\tau_0, \dots, \tau_{d-1})$, with the constraint

$$\sum_{j=0}^{d-1} ((d-j)\tau_j) \leq d. \tag{6}$$

Next, we are going to explain why this constraint is sufficient. For convenience, we denote the dimension of $\nu_{X(c)}$ as $d(c)$, and the dimension of the contributor, ν_{X_i} , as d_i , namely, $d_i = \dim(\nu_{X_i})$. Generally, the dimension of the intersection of two Voronoi cells, ν_X and ν_Y , is $\dim(\nu_X \cap \nu_Y) = \dim(\nu_X) + \dim(\nu_Y) - d$. The Voronoi cell, $\nu_{X(c)}$, which is the intersection of n Voronoi cells, has the dimension

$$d(c) = \sum_{i=1}^n d_i - (n-1)d. \tag{7}$$

We can get the Constraint (6) by substituting $d(c) \geq 0$, $\sum_{i=1}^n d_i = \sum_{j=0}^d (j\tau_j)$, and $n = \sum_{j=0}^d \tau_j$ into the Inequality (7).

To list out all the possible tuples in \mathbb{R}^d , we firstly list out all the possible values of τ_0 under the Constraint (6). We get all the possible values of τ_1 for each value of τ_0 . Repeating this progress from τ_1 to τ_{d-1} under the Constraint (6), we can enumerate all the possible tuples. For example, there are seven types of Voronoi cells in \mathbb{R}^3 , under the constraint $3\tau_0 + 2\tau_1 + \tau_2 \leq 2$. The tuples,

$(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$ and $(0, 0, 0)$, represent the four simplicial types, Voronoi vertex, edge, polygon, and polytope, respectively. The tuples, $(0, 0, 2)$, $(0, 0, 3)$, and $(0, 1, 1)$ represent three degenerate types.

4 Algorithm for Computing Intermediate Complexes

An intermediate shape, $\text{body}(B(c))$, is immediate once the intermediate Voronoi complex, $V(c)$, is determined. In this section, we introduce the algorithm to compute $V(c)$.

An intuitive yet time consuming approach is to compute the Voronoi complex of $B(c)$. Assuming $\text{card}(B_i) \leq m, i = 1..n$, the cardinality of $B(c)$ is no more than m^n . Therefore, the time for computing the Voronoi complex of $B(c)$ directly is

$$O(nm^n \log m + m^{n \times \lceil d/2 \rceil}) \tag{8}$$

[1, 3, 6].

On the other hand, according to Theorem 1, we can compute $V(c)$ by superimposing the signed Voronoi complexes of the reference shapes. We prove that when $d > 2$, computing $V(c)$ by superimposing is faster than computing it from $B(c)$ directly.

Assuming the value of c is fixed, we simplify the notations $V_{B_i}^{\text{Sign}(\gamma_i)}$ to V_{B_i} , and $\nu_{X_i}^{\text{Sign}(\gamma_i)}$ to ν_{X_i} in the rest of the paper.

Superimposing Two Voronoi Complexes. Let V_0 and V_1 be two intermediate Voronoi complexes. We construct their superimposition, $V_{0,1}$ by computing each of its Voronoi region $\nu_{i,j} = \nu_i \cap \nu_j$, for $\nu_i \in V_0$ and $\nu_j \in V_1$. We can compute $V_{0,1}$ by brute force, namely, testing all the possible pairs of ν_i and ν_j for non-empty intersection. This can be achieved by linear programming algorithms.

However, we can improve the construction by a breath-first search manner. Given one Voronoi region $\nu_{i,j} \in V_{0,1}$, we can compute the neighbors of $\nu_{i,j}$ by considering all neighbors of ν_i and ν_j in V_0 and V_1 respectively. The time for this breadth-first search algorithm is output sensitive. The worst case is that we tested all the possible pairs of Voronoi cells, namely m^2 pairs. Testing whether two Voronoi regions have non-empty intersection needs $O(d!m)$ time by the linear programming method [5]. Thus, the total time complexity of superimposing V_0 and V_1 is $O(d!m^3)$.

Superimposing n Voronoi Complexes. Next we introduce the algorithm of computing $V(c)$ by superimposing n Voronoi complexes, which is based on the algorithm of superimposing two Voronoi complexes.

For convenience, let $k = \lceil \log_2(n) \rceil$ and $\eta = 2^{k-1}$ and we have $\eta < n \leq 2\eta$. We divide the way of superimposing n Voronoi complexes into three cases, namely, $\eta < n \leq 9\eta/8$, $9\eta/8 < n \leq 3\eta/2$, and $3\eta/2 < n \leq 2\eta$. We prove that when $d > 2$, our algorithm can achieve better efficiency than computing the Voronoi complex of $B(c)$ directly.

We will prove the case for $3\eta/2 < n \leq 2\eta$ first. We superimpose the n Voronoi complexes according to a binary superimposing tree. See Figure 5. In the first

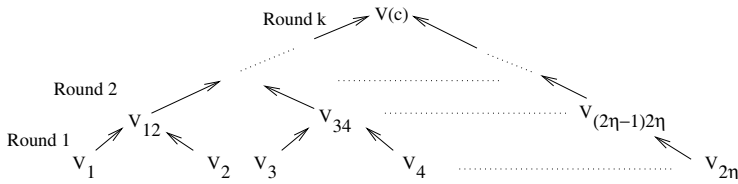


Fig. 5. Superimposing n Voronoi complexes when $3\eta/2 < n \leq 2\eta$

round, we superimpose V_{2j-1} and V_{2j} for $j = 1.. \eta$. The time complexity for this round is $O(d!2^{k-1}m^3)$. In the second round, we superimpose $V_{(4j-3)(4j-2)}$ and $V_{(4j-1)(4j)}$ for $j = 1.. \eta/2$. In either $V_{(4j-3)(4j-2)}$ or $V_{(4j-1)(4j)}$, there are no more than m^2 Voronoi regions. Thus, the time complexity for the second round is $O(d!2^{k-2}m^{3 \times 2})$. Generally, for the i -th round, the time complexity is $O(d!2^{k-i}m^{3 \times 2^{i-1}})$, $i = 1..k$. After k rounds of such superimposition, we can get $V(c)$. Therefore, the total time complexity to superimpose the n Voronoi complexes, V_1, V_2, \dots and V_n , is

$$O\left(\sum_{i=1}^k (d!2^{k-i}m^{3 \times 2^{i-1}})\right).$$

Since m is greater than 2 in practice, we have $(d!2^{k-i_1}m^{3 \times 2^{i_1-1}}) < (d!2^{k-i_2}m^{3 \times 2^{i_2-1}})$ for any $i_1 < i_2$. Thus, the total time complexity is equivalent to $O(kd!m^{3 \times 2^{k-1}})$. For a fixed d , we can ignore the $d!$ and only consider the time complexity as $O(km^{3 \times 2^{k-1}})$. Adding in the time of computing V_1, \dots, V_n , the overall time complexity to compute $V(c)$ is

$$O(km^{3 \times 2^{k-1}} + nm \log m + nm^{\lceil d/2 \rceil}). \tag{9}$$

When $d > 2$ and $n > 1$, the time complexity of the intuitive way in Equation 8 is dominated by $O(m^{n \times \lceil d/2 \rceil})$, and the time complexity of our algorithm in Equation 9, is dominated by $O(km^{3 \times 2^{k-1}} + nm^{\lceil d/2 \rceil})$. In practice, we can assume that n, k , and d are smaller than m . Under such assumption, our algorithm has better efficiency if we can show that $m^{3 \times 2^{k-1}} < m^{n \lceil d/2 \rceil}$. Since $n > 3\eta/2$ and $d > 2$, we have

$$m^{3 \times 2^{k-1}} = m^{3 \times \eta} < m^{2n} \leq m^{n \lceil d/2 \rceil},$$

as required.

When $9\eta/8 < n \leq 3\eta/2$ and $\eta < n \leq 9\eta/8$, we can build the superimposing tree in two similar ways, both of which has better efficiency than the intuitive way.

5 Conclusion

In this paper, we designed a new algorithm to compute the Voronoi complex of an intermediate shape basing on Theorem 1. We prove that when $d > 2$, our algorithm is faster than computing the Voronoi complex from $B(c)$ directly. Moreover, since all the intermediate shapes share finitely many common Voronoi complexes, we are able to compute intermediate shapes in real time, by reusing the generated common Voronoi complexes. This makes it possible to generate, visualize, and customize shape deformations.

About the future direction, we want to save the time complexity of superimposing n Voronoi complexes by rearranging the order of the pairwise superimpositions. We may solve this problem with dynamic programming. The goal is to change the superimposing order so that the cardinalities of the intermediate superimposing results are as small as possible.

References

- [1] CHAZELLE, B. An optimal convex hull algorithm in any fixed dimension. In *Discrete Comput. Geom.* (1993), pp. 10:377–409.
- [2] CHENG, H., EDELSBRUNNER, H., AND FU, P. Shape Space from Deformation. *Proc. 6th Pacific Conf.. Comput. Graphics Appl.* (1998), 104–113.
- [3] CLARKSON, K. L., AND SHOR, P. W. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry* 4, 1 (1989), 387–421.
- [4] EDELSBRUNNER, H. Deformable Smooth Surface Design. *Discrete Comput. Geom* 21 (1999), 87–115.
- [5] SEIDEL, R. Linear programming and convex hulls made easy. In *Proc. 6th Annu ACM Sympos. Coput. Geom.* (1990), pp. 211–215.
- [6] SEIDEL, R. Small-dimensional linear programming and convex hulls made easy. In *Discrete Comput. Geom.* (1991), pp. 6:423–434.

On Partial Lifting and the Elliptic Curve Discrete Logarithm Problem (Extended Abstract)

Qi Cheng^{1,*} and Ming-Deh Huang^{2,**}

¹ School of Computer Science,
The University of Oklahoma,
Norman, OK 73019, USA
qcheng@cs.ou.edu

² Computer Science Department,
University of Southern California,
Los Angeles, CA 90089
huang@cs.usc.edu

Abstract. It has been suggested that a major obstacle in finding an index calculus attack on the elliptic curve discrete logarithm problem lies in the difficulty of lifting points from elliptic curves over finite fields to global fields. We explore the possibility of circumventing the problem of explicitly lifting points by investigating whether partial information about the lifting would be sufficient for solving the elliptic curve discrete logarithm problem. Along this line, we show that the elliptic curve discrete logarithm problem can be reduced to three partial lifting problems. Our reductions run in random polynomial time assuming certain conjectures are true. These conjectures are based on some well-known and widely accepted conjectures concerning the expected ranks of elliptic curves over the rationals.

Keywords: Elliptic curve cryptosystem, discrete logarithm, partial lifting.

1 Introduction

The discrete logarithm problem over elliptic curves is a natural analog of the discrete logarithm problem over finite fields. It is the basis of elliptic curve cryptosystems proposed independently by Koblitz and Miller [9, 10]. Steady progress has been made in constructing better and more sophisticated, albeit subexponential time algorithms for the discrete logarithm problem over finite fields. In contrast, no subexponential attacks have been found for the elliptic curve discrete logarithm problem except in very special cases [4, 5, 13, 14]. Consequently

* This research is supported in part by NSF career award CCR-0237845.

** This research is supported in part by NSF grant CCR-9820778.

elliptic curve cryptosystems have attracted considerable attention, especially in cryptographic applications where key length needs to be kept to the minimal.

Most subexponential algorithms for discrete logarithms over finite fields have been based on the *lifting method* (see [20] for a survey). This method involves lifting elements from a finite field to a global field to take advantage of the arithmetic structures in the global field. The lifting of elements is simple and straightforward. For example in the case of a finite prime field \mathbf{F}_p , an element $a \bmod p \in \mathbf{F}_p$ with $0 < a < p$ is simply lifted to $a \in \mathbf{Z}$. However extending this method to the elliptic curve discrete logarithm problem seems to be difficult. It has been suggested [10] that a major obstacle in finding an index calculus attack on the elliptic curve discrete logarithm problem lies in the difficulty of lifting points from elliptic curves over finite fields to global fields. The reason behind such difficulty is that elliptic curves over \mathbf{Q} usually have very small rank – at least heuristically and practically speaking. As a result rational points with reasonably bounded heights are severely limited in number, rendering the lifting problem difficult. (See the next section for an illustration, and [7, 8, 19] for more in-depth discussion).

In this paper we explore the possibility of circumventing the problem of explicitly lifting points by investigating whether partial information about the lifting would be sufficient for solving the elliptic curve discrete logarithm problem. We show that the elliptic curve discrete logarithm problem can be reduced to three partial lifting problems. These partial lifting problems have the same basic setup as the explicit lifting problem, namely, an elliptic curve E/\mathbf{F}_p a nonzero point $S \in E(\mathbf{F}_p)$, an elliptic curve \mathcal{E}/\mathbf{Q} having E as its good reduction mod p , and $X \in \mathcal{E}(\mathbf{Q})$ which reduces to $S \bmod p$. Moreover we assume that $E(\mathbf{F}_p)$ is cyclic of prime order, so that every point in $E(\mathbf{F}_p)$ is liftable to $\mathcal{E}(\mathbf{Q})$. For the first partial lifting problem, we are given $T \in E(\mathbf{F}_p)$, a height bound $h \in \mathbf{Q}$, and the goal is to decide whether T can be lifted to a point in $\mathcal{E}(\mathbf{Q})$ of height bounded by h . We call this problem the *height decision problem*. Note that if the height bound h is around p , it may take exponential time just to write down a lift of T . However in the height decision problem all that we ask is whether such a lift exists or not. For the second partial lifting problem, we are given $T \in E(\mathbf{F}_p)$ and a prime r , and the goal is to find the reduction mod r of any one lift of T to $\mathcal{E}(\mathbf{Q})$. We call this problem the *reduction problem*. Note that a lift of T can again be large, but its reduction modulo r has length $O(\log r)$. For the third partial lifting problem, we are given $T \in E(\mathbf{F}_p)$, and the goal is to construct a straight-line description of any lift of T to $\mathcal{E}(\mathbf{Q})$. We call this problem the *straight-line problem*. (As will be observed in the next section that a small straight-line description of a lift of T usually does exist).

Our reductions run in random polynomial time assuming certain conjectures are true. These conjectures are based on some well-known and widely accepted conjectures concerning the expected ranks of elliptic curves over the rationals. They are stated explicitly in the next section. Our results lead to the following open question: can partial information on lifting as described above be extracted in subexponential time? An affirmative answer will lead to a subexponential algo-

rithm for the the elliptic curve discrete logarithm problem. On the other hand, should the elliptic curve discrete logarithm problem admit no subexponential time attack, then our results suggest that gaining partial information about lifting would be just as hard.

2 Statements of Results

An elliptic curve is a smooth cubic algebraic curve. Let k be a field. An elliptic curve over k can be given as an equation of the form:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_i \in k, 1 \leq i \leq 6$. Denote by $E(k)$ the set of points $(x, y) \in k^2$ that satisfy this equation, along with a point O at infinity. If the characteristic of k is neither 2 nor 3, we may assume that the elliptic curve is given by an equation of the form

$$y^2 = x^3 + ax + b, \quad a, b \in k$$

The discriminant of this curve is defined as the discriminant of polynomial $x^3 + ax + b$, which is $-4a^3 - 27b^2$. The curve is smooth iff its discriminant is not zero.

The set $E(k)$ forms an additive group. Let $S, T \in E(k)$ be two points on the curve. The discrete logarithm of T with base S is an integer m such that $T = mS$, if such an m exists.

A lot of theoretical and experimental evidence shows that most elliptic curves would have as small a rank as allowed by the sign of their functional equations[2, 3]. In particular most elliptic curves over the rationals with a rational point of infinite order are expected to be either of rank 1 or 2. Parts of our proofs are based on such a heuristic assumption. An explicit statement of the assumption sufficient for our purposes is given below.

Let $E : y^2 = x^3 + ax + b$ be an elliptic curve over $Z/(n)$. Let P be a point on the E all of whose coordinates are in $(Z/(n))^*$ and let X be the natural lift of P to \mathbf{Q} . Consider the family of curves over \mathbf{Q}

$$\mathcal{F}(E, X) = \{\mathcal{E}(\alpha, \beta) \mid |\alpha|, |\beta| \leq n^2, \mathcal{E} \text{ pass } X \text{ and } \mathcal{E} \text{ reduce to } E \text{ modulo } n\}$$

where $\mathcal{E}(\alpha, \beta)$ is defined by the equation $y^2 = x^3 + \alpha x + \beta$. Then for sufficiently large n , a random curve in the family has rank 1 with probability greater than some constant.

In light of the general heuristic assumption, it would be reasonable to expect that a random curve in $\mathcal{F}(E, X)$ has rank 1 with probability around 1/2, and rank 2 with probability around 1/2.

Now to illustrate the difficulty of lifting points, let us consider an elliptic curve E/\mathbf{F}_p with a nonzero point $S \in E(\mathbf{F}_p)$ which generates a cyclic group $\langle S \rangle$ of large prime order r . It is not difficult to construct an elliptic curve \mathcal{E}/\mathbf{Q} with E as its reduction at p , and $X \in \mathcal{E}(\mathbf{Q})$ which reduces to S modulo p .

In practical terms $\mathcal{E}(\mathbf{Q})$ would most likely be of rank one or two. For simplicity suppose it is of rank one. Now suppose we want to lift a point T in the group $\langle S \rangle$ to a point in $\mathcal{E}(\mathbf{Q})$. Suppose $\mathcal{E}(\mathbf{Q})$ has trivial torsion part and, to our advantage, S is the image of the a generator X of $\mathcal{E}(\mathbf{Q})$ upon reduction mod p . Then for $m < r$, mX is the lift of $T = mS$ of minimum possible canonical height. However $\hat{h}(mX) = m^2\hat{h}(X)$ (where \hat{h} denotes the canonical height function), which is $\Omega(p^2)$ in the worst case. Therefore it is infeasible even to write down the coordinates of the lifting points of T .

$$\begin{array}{ccc} Y = m X & \in & \mathcal{E}(\mathbf{Q}) \\ \uparrow & & \uparrow \\ T = m S & \in & E(\mathbf{F}_p) \end{array}$$

On the other hand, though the coordinates of Y are huge they actually have very short straight-line description essentially because multiplication of a point by m can be performed in $O(\log m)$ additions of points. So instead of trying to lift points explicitly we consider three partial lifting problems whose formal definitions are given below together with theorems concerning their relation to the elliptic curve discrete logarithm problem.

The Height Decision Problem:

Input: $p, E, S, \mathcal{E}, X, T$ and h where p is a prime, E/\mathbf{F}_p is an elliptic curve such that $E(\mathbf{F}_p)$ is cyclic of prime order, S is a nonzero point in $E(\mathbf{F}_p)$, \mathcal{E}/\mathbf{Q} is an elliptic curve having E as its good reduction mod p , $X \in \mathcal{E}(\mathbf{Q})$ which reduces to S mod p , $T \in E(\mathbf{F}_p)$ and $h \in \mathbf{Q}$ (a height bound).

Output: "Yes" if T can be lifted to a point in $\mathcal{E}(\mathbf{Q})$ of height bounded by h , and "no" otherwise.

Theorem 1.

The Shifting Prime Problem:

Input: p, r, \mathcal{E}, B and P where p and r are prime numbers, \mathcal{E}/\mathbf{Q} is an elliptic curve, B is a nontorsion rational point on the curve, P is a point on E_1 where E_1 is the reduction of \mathcal{E} modulo p . We require that B doesn't reduce to O modulo p or r .

Output: A point $R \in \mathbf{P}^2(\mathbf{F}_r)$ which is the reduction of $X \in \mathcal{E}(\mathbf{Q})$ modulo r , where X is any lift of P to \mathbf{Q} .

Theorem 2.

The Straight-Line Description Problem:

Input: p, r, \mathcal{E}, B and P where p is a prime number, \mathcal{E}/\mathbf{Q} is an elliptic curve, B is a nontorsion rational point on the curve, P is a point on E_1 where E_1 is the reduction of \mathcal{E} modulo p . We require that B doesn't reduce to O modulo p .

Output: Straight-line programs of length polynomial in the size of input for the projective coordinates of any lift of P on \mathcal{E} .

Theorem 3.

3 Reduction to the Height Decision Problem

In this section, we prove Theorem 1 by demonstrating a random polynomial time reduction from the elliptic curve discrete logarithm problem to the height decision problem.

Suppose for the rest of this section that p is a prime larger than 3 and E is an elliptic curve defined over \mathbf{F}_p given by $y^2 = x^3 + ax + b$ with $a, b \in \mathbf{F}_p$, $E(\mathbf{F}_p)$ is cyclic of prime order r , and $S, T \in \mathcal{E}(\mathbf{F}_p)$. In the discrete logarithm problem we are to find m so that $T = mS$ on $E(\mathbf{F}_p)$.

First we observe that if p is sufficiently large, then any lift \mathcal{E} of E with good reduction cannot have any nontrivial rational torsion point. Indeed by a result of Mazur [11, 12], we know that $\mathcal{E}(\mathbf{Q})$ has at most 16 torsion points. Since the torsion subgroup injects into $E(\mathbf{F}_p)$ and since the order of $E(\mathbf{F}_p)$ is prime, it follows that the torsion part of $\mathcal{E}(\mathbf{Q})$ must be trivial. From now on we assume that p is large enough.

Let $X = (x_0, y_0)$ be the natural lift of S to \mathbf{Z} .

In the first step of our reduction we construct a lift of E to some \mathcal{E}/\mathbf{Q} passing through X , given by $y^2 = x^3 + \alpha x + \beta$ with $\alpha, \beta \in \mathbf{Z}$, and $(\alpha, \beta) = 1$. To this end we choose $\alpha = a + ip$ with $|i| < p$ so that $(\alpha, y_0^2 - x_0^3) = 1$, then set $\beta = y_0^2 - x_0^3 - \alpha x_0$. It is easy to see that \mathcal{E} passes through X . Moreover one can show that a substantial fraction of integers i satisfies the requirement, so that $(\alpha, \beta) = 1$. It then follows from the results in [16, 17, 18] that one can compute $h_0 \in \mathbf{Q}$ in time polynomial in $\log p$ so that $|\hat{h}(X) - h_0| < \frac{1}{r^2}$.

Since $X \in \mathcal{E}(\mathbf{Q})$ is non-torsion, $\mathcal{E}(\mathbf{Q})$ is of rank at least one. Moreover the reduction map from $\mathcal{E}(\mathbf{Q})$ to $E(\mathbf{F}_p)$ is surjective, since $S = X \bmod p$ and $E(\mathbf{F}_p)$ is cyclic of prime order. Therefore every point in $E(\mathbf{F}_p)$ has a lift to $\mathcal{E}(\mathbf{Q})$. In particular, T has a lift of the form mX .

Suppose $\mathcal{E}(\mathbf{Q})$ is of rank one and G is a generator of $\mathcal{E}(\mathbf{Q})$. (We will not actually compute the rank nor a generator.) Then $X = lG$ for some $l \in \mathbf{Z}$, and if $nG \bmod p = T$ ($n < r$), lT has a lift nX . From l and n , the discrete logarithm problem is solved, since upon reduction we get $lT = nS$.

Since $X = lG$, $\hat{h}(X) = l^2\hat{h}(G)$. Now by construction, $\hat{h}(X) = (\log p)^{O(1)}$. On the other hand, by Lang’s conjecture, $\hat{h}(G) > c \log \Delta$ where c is an absolute constant and Δ is the discriminant of \mathcal{E} . It follows that $l = (\log p)^{O(1)}$.

By the result in [16, 17, 18], we have $|2\hat{h}(Y) - h(Y)| < c$ for $Y \in \mathcal{E}(\mathbf{Q})$, where c is a constant independent of Y and \mathcal{E} . In particular for positive integers $i < r$,

$$|\frac{1}{2}h(iX) - i^2h_0| \leq |\frac{1}{2}h(iX) - \hat{h}(iX)| + |\hat{h}(iX) - i^2h_0| < c/2 + 1.$$

Set $c' = c/2 + 1$. Then it follows that if $\frac{1}{2}h(nX) < i^2h_0$, then $n \leq i + c'$. This implies that using a binary search technique beginning with the query asking if lT is liftable to a point of height no greater than $2r^2h_0$, we can determine n within $O(1)$ in $O(\log p)$ queries.

Consequently, for the constructed lift \mathcal{E} and each value of l up to $(\log p)^{O(1)}$, we will attempt to find an $n < r$ so that lT has a lift to $\mathcal{E}(\mathbf{Q})$ of the form nX . When we succeed to find such n for an l , we then verify if $lT = nS$ and if so, the discrete logarithm is solved. If we fail for all possible values of l , then it must be the case that the rank of $\mathcal{E}(\mathbf{Q})$ is greater than one. In that case we construct another random lift and apply the same procedure all over again. By our heuristic assumption, a randomly constructed $\mathcal{E}(\mathbf{Q})$ has rank one with probability about $1/2$, hence we expect to succeed within two trials. Hence Theorem 1 follows.

4 The Shifting Prime Problem

In this section, we show that the shifting prime problem is equivalent to the discrete logarithm problem on elliptic curves. The main idea in the proof is to lift an elliptic curve E/\mathbf{F}_p to an elliptic curve \mathcal{E}/\mathbf{Q} of rank one with additive reduction modulo r , where r is the prime order of $E(\mathbf{F}_p)$. We demonstrate that, with the help of an oracle for the shifting prime problem, the discrete logarithm problem on $E(\mathbf{F}_p)$ can be shifted over to the group of nonsingular \mathbf{F}_r -points on $\mathcal{E} \bmod r$, which is isomorphic to the addition group of F_r , and the corresponding discrete logarithm problem is easy to solve.

First we review some facts about additive reduction. Let $\mathcal{E}(\mathbf{Q}) : y^2 = x^3 + ax + b$, $a, b \in \mathbf{Z}$, be an elliptic curve over \mathbf{Q} . It is possible that when modulo some prime p , the reduction curve E/\mathbf{F}_p is not smooth anymore. If E has a cusp, we say that E is an additive reduction of \mathcal{E} at p .

Let $E^{(ns)}$ be the set of non-singular points on E , we can define “addition” on $E^{(ns)}$ in very much the same way as in the smooth case [15]. Moreover,

$$E^{(ns)} \cong F_p^+$$

by sending (x, y) to $\frac{x}{y}$. (Note that since (x, y) is not a singular point, $y \neq 0$.) Hence the discrete logarithm problem on $E^{(ns)}$ is easy to solve.

For example, if \mathcal{E} is defined by

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbf{Z}$. $p \neq 2, 3$. \mathcal{E} has additive reduction at p if and only if $p|a$ and $p|b$.

Let $\mathcal{E}(\mathbf{Q})$ be an elliptic curve with rank 1 and with no rational torsion point other than O , and let G be the generator for $\mathcal{E}(\mathbf{Q})$. Let E_1/\mathbf{F}_p be the reduction of \mathcal{E} modulo p . Suppose the order of $E_1(\mathbf{F}_p)$ is a prime r . Moreover suppose \mathcal{E} has additive reduction mod r , and let E_2/\mathbf{F}_r be the resulting curve. Let G_1 and G_2 be the reduction of G on E_1 and E_2 respectively. Moreover suppose that G_1 and G_2 are neither points at infinity nor singular. It follows that all the points on $\mathcal{E}(\mathbf{Q})$ reduce to smooth points on E_1 and E_2 . Let $E_2^{(ns)}$ denote the set of non-

singular points on the curve E_2 . Then $\mathcal{E}(\mathbf{Q}) \rightarrow E_1(\mathbf{F}_p)$ and $E(\mathbf{Q}) \rightarrow E_2^{(ns)}(\mathbf{F}_r)$ are group homomorphisms.

$$\begin{array}{ccc}
 & \mathcal{E}(\mathbf{Q}) & \\
 \swarrow & & \searrow \\
 E_1(\mathbf{F}_p) & \xrightarrow{\psi} & E_2^{(ns)}(\mathbf{F}_r)
 \end{array}$$

Since $E_1(\mathbf{F}_p)$ and $E_2^{(ns)}(\mathbf{F}_r)$ have same order, there is a well-defined isomorphism ψ between $E_1(\mathbf{F}_p)$ and $E_2^{(ns)}(\mathbf{F}_r)$ determined by

$$\psi(iG_1) = iG_2.$$

Suppose $T, S \in E(\mathbf{F}_p)$, $T = mS$ and we want to find m . Certainly $\psi(T) = m\psi(S)$. If we can solve the shifting prime problem efficiently, we will get $\psi(T), \psi(S)$. Hence we have reduced the discrete logarithm problem in $E_1(\mathbf{F}_p)$ to the discrete logarithm problem in $E_2^{(ns)}(\mathbf{F}_r)$. Since E_2 is an additive reduction of \mathcal{E} , the discrete logarithm on E_2 is simply division in finite fields.

Suppose $E_1 : y^2 = x^3 + ax + b$ is an elliptic curve over F_p with r points. S and T are the input points for the discrete logarithm. Assume that $r > 3$ is a prime. The reduction algorithm is as follows:

Algorithm 1. $E_2 : y^2 = x^3 + ax + b$ over F_r

$E_1 \dots E_2 \dots E_3 \dots Z/(pr)$
 $E_3 \dots \mathcal{E} \dots \mathbf{Q} \dots X \dots X$
 $O \dots E_1 \dots E_2 \dots B \dots E_3 \dots B$
 $S' \dots T' \dots S \dots T \dots r$
 $S' \dots E_2 \dots T \dots m'$
 $T = m'S \dots m'$

Step 2 can be done very efficiently. By Conjecture 1 we expect the curve constructed in step 3 to be of rank one with reasonable probability. With the same probability, we will get the correct discrete logarithm of T , once the shifting prime problem is solved. Thus Algorithm 1 reduces the elliptic curve discrete logarithm problem to the shifting prime problem in random polynomial time, and Theorem 2 follows.

As in the previous reduction, we note that the reduction here is a Turing reduction. Since discrete logarithm is easily to check for correctness, there is no need to verify if the lifting curve is of rank one.

5 Straight-Line Program for the Lifting Points

In this section, we will derive theorem 3 from theorem 2. We first give a formal definition of a straight-line program.

Definition 1. $m(m_1, m_2, \dots, m_n)$
 $z \leftarrow x\alpha y$
 $\alpha \in \{*, +, -\}$
 $m(m_1, m_2, \dots, m_n)$

In some cases, a straight-line program is a compact description of an integer. It can represent a huge number in small length. For example, m^n has a straight-line program of length $(\log mn)^{O(1)}$. It is an important open problem whether $n!$ has a short straight-line program.

It seems hard to compute with straight-line programs. For example, given straight-line programs for the integers x and y , it is not a trivial problem to decide whether $x = y$. However, from a straight-line program of integer i , we can read out the reduction of i modulo any prime p , by performing every step of the straight-line program modulo p . Similarly, if we have the straight-line program for the coordinates of a global point $P = (x : y : z) \in \mathbf{P}^2(\mathbf{Q})$, we usually can calculate the reduction of P at p for any given prime p .

However, there is some subtlety here. Let x, y, z be the integers output by a straight-line program. If $p \nmid \gcd(x, y, z)$, we can compute the reduction of $(x : y : z)$ at p efficiently. If x, y, z share a lot of p 's, after executing the straight-line program modulo p (or p^i for i small), we get the point $(0 : 0 : 0)$, which is not well-defined in the projective space. This motivates us to define properly reduced coordinates. Without loss of generality, let \mathcal{E}/\mathbf{Q} be an elliptic curve of rank one with no rational torsion point other than O .

Definition 2. $(x' : y' : z') = m(x : y : z) \in \mathcal{E}(\mathbf{Q})$
 $(x : y : z) \in \mathcal{E}(\mathbf{Q})$
 $p \nmid z'$
 $(x : y : z) \in \mathcal{E}(\mathbf{Q})$
 $p \nmid m$
 x', y', z'
properly reduced

Let E be the reduction of \mathcal{E} at p . Assume that every point on E is liftable to \mathcal{E} . Let P, r be the remaining input of shifting prime problem. Given an algorithm which can solve the straight-line description problem with properly reduced output, we can lift P to X , which is represented by a straight-line program of length polynomial in the size of the input. Then making use of the fact that the output of the straight-line program is properly reduced, we can compute X modulo r for any prime r , hence provide answers to the shifting prime queries. This means that we have reduction from shifting prime problem to straight-line description problem. Thus Theorem 3 follows from theorem 2.

Theorem 3 raises the question of the existence of a “short” straight-line program for properly reduced coordinates of a point on the elliptic curve. We give an affirmative answer to this question.

Proposition 1. $m, x, y, z, a, b \in \mathbf{Z}$
 $\mathcal{E} : y^2 = x^3 + ax + b$
 $\gcd(x, y, z) = 1$
 $(x : y : z)$

$$\log^{O(1)}(xyzmab), \quad (x' : y' : z') = m(x : y : z) \in \mathcal{E}(\mathbf{Q}) \quad x', y', z' \in \mathbf{Z}$$

Proof: First, we consider the explicit formulas for addition of two points and for doubling a point.

Let $(x_1 : y_1 : z_1)$ and $(x_2 : y_2 : z_2)$ be two points on an elliptic curve $\mathcal{E} : y^2 = x^3 + ax + b, x_1, y_2, z_1, x_2, y_2, z_2 \in \mathbf{Z}$. Assume that $GCD(x_1, y_1, z_1) = GCD(x_2, y_2, z_2) = 1$. If $x_1z_2 \neq x_2z_1$, then according to the addition law on elliptic curves, $(x_1 : y_1 : z_1) + (x_2 : y_2 : z_2) = (x_3 : y_3 : z_3)$, where

$$\begin{aligned} x_3 &= z_1z_2(y_2z_1 - y_1z_2)^2(x_2z_1 - x_1z_2) - (x_1z_2 + x_2z_1)(x_2z_1 - x_1z_2)^3 \\ y_3 &= (y_2z_1 - y_1z_2)(z_1z_2(y_2z_1 - y_1z_2)^2 - (x_1z_2 + x_2z_1)(x_2z_1 - x_1z_2)^2) \\ &\quad - z_1z_2(y_1x_2 - y_2x_1)(x_2z_1 - x_1z_2)^2 \\ z_3 &= z_1z_2(x_2z_1 - x_1z_2)^3 \end{aligned}$$

Lemma 1. $p \neq 2, 3 \quad (x_1 : y_1 : z_1) \neq (x_2 : y_2 : z_2) \pmod p \quad (x_1 : y_1 : z_1) \not\equiv -(x_2 : y_2 : z_2) \pmod p \quad p \nmid z_3$

If $(x_1 : y_1 : z_1) = (x_2 : y_2 : z_2)$ and $y_1z_1 \neq 0$, then

$$\begin{aligned} x_3 &= 2y_1z_1(3x_1^2 + az_1^2) - 16x_1y_1^3z_1^2 \\ y_3 &= -(3x_1^2 + az_1^2)((3x_1^2 + az_1^2) - 8x_1y_1^2z_1) - 4y_1^2z_1(x_1^3 + ax_1z_1^2 + 2bz_1^3) \\ z_3 &= 8y_1^3z_1^3 \end{aligned}$$

Lemma 2. $p \neq 2, 3 \quad (x_1, y_1, z_1) \neq (x_2, y_2, z_2) \pmod p \quad O \pmod p \quad p \nmid z_3$

Let $m = \sum_{i=1}^a 2^{e_i}, e_1 < e_2 < \dots < e_a$, then

$$m(x : y : z) = \sum_{i=1}^a 2^{e_i}(x : y : z).$$

When we apply the squaring technique to write straight-line program for $m(x : y : z)$, we first use the doubling formula to compute $2^{e_1}(x : y : z), 2^{e_2}(x : y : z), \dots, 2^{e_a}(x : y : z)$. Then we use formula for addition of two different points to sum them up. It is not possible that some of the intermediate points will be equal, as long as $(x : y : z)$ is not a torsion point. This concludes the proof of proposition 1.

Let $\mathcal{E}/\mathbf{Q} : y^2 = x^3 + ax + b, a, b \in \mathbf{Z}$, be an elliptic curve with rank 1. Every liftable point on its reduction curve has a short straight-line program. More precisely, we have

Corollary 1. $E \text{ is a } p\text{-torsion point} \quad \mathcal{E} \text{ is } p\text{-torsion} \quad p \mid Q \in E/\mathbf{F}_p \quad \mathcal{E}(\mathbf{Q}) \text{ is } p\text{-torsion} \quad X \in \mathcal{E}(\mathbf{Q}) \quad \log^{O(1)}(abp)$

References

1. L.M. Adleman, A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In FOCS 79.
2. A. Brumer, The average rank of elliptic curves I, *Invent. Math.* 109(1992), 445-472.
3. J.E. Cremona, *Algorithms for modular elliptic curves*, Cambridge University Press 1992.
4. G. Frey, Applications of arithmetical geometry to cryptographic constructions, Proceedings of the Fifth International Conference on Finite Fields and Applications, Springer-Verlag, 2001.
5. P. Gaudry, F. Hess and N. Smart, Constructive and destructive facets of Weil descent on elliptic curves, *J. of Cryptology* 15 (2002), 19-46.
6. M. Jacobson, A. Menezes and A. Stein, Solving elliptic curve discrete logarithm problems using Weil descent, *Journal of the Ramanujan Mathematical Society*, 16 (2001), 231-260.
7. M.-D. Huang, K.L. Kueh and K.-S. Tan, Lifting elliptic curves and solving the elliptic curve discrete logarithm problem, *Proc. Algorithmic Number Theory Symp. (ANTS IV)*, (2000), 377-384.
8. M.J. Jacobson, N. Koblitz, J.H. Silverman, A. Stein, and E. Teske, Analysis of the Xedni Calculus Attack, *Designs, Codes and Cryptography*, 20, 2000.
9. N. Koblitz, elliptic curve cryptosystems, *Mathematics of Computation*, 48(1987), 203-209.
10. V. Miller, Uses of elliptic curves in cryptography, *Advances in Cryptology: Proceedings of Crypto'85*, Lecture Notes in Computer Science, 218(1986), Springer-Verlag, 417-426.
11. B. Mazur, Modular curves and the Eisenstein ideal, *IHES publi. Math.* 47 (1977).
12. B. Mazur, Rational isogenies of prime degree, *Invent. Math.* 44(1978).
13. A. Menezes and M. Qu, Analysis of the Weil descent attack of Gaudry, Hess and Smart, CT-RSA 2001, LNCS 2020.
14. A. Menezes, E. Teske and A. Weng, Weak fields for ECC, Topics in Cryptology – CT-RSA 2004, Lecture Notes in Computer Science, 2964 (2004), 366-386.
15. J.H. Silverman, *The arithmetic of elliptic curves*, Springer-Verlag, 1986.
16. J.H. Silverman, Computing Heights on elliptic curves, *Mathematics of computation*, Vol 51, 1988.
17. J.H. Silverman, the difference between the Weil height and the canonical height on elliptic curves, *Mathematics of computation*, Vol 55, 1990.
18. J.H. Silverman, Computing canonical heights with little(or no) factorization, *Mathematics of computation*, Vol 66, 1997.
19. J.H. Silverman and J. Suzuki, Elliptic curve discrete logarithms and the index calculus, *Advances in Cryptology-Asiacrypt'98*, Spring-Verlag, 1998, 110-125.
20. O. Schirokauer, D. Weber and Th. Denny, Discrete logarithms: The effectiveness of the index calculus method, ANTS II, LNCS 11122, Spring-Verlag, 1996.

Guarding Art Galleries by Guarding Witnesses (Extended Abstract)

Kyung-Yong Chwa¹, Byung-Cheol Jo², Christian Knauer³, Esther Moet⁴,
René van Oostrum⁴, and Chan-Su Shin⁵

¹ Department of Computer Science, KAIST, Daejeon, Korea
kychwa@tclab.kaist.ac.kr

² Taff System, Co. Ltd., Seoul, Korea
mrjo@taff.co.kr

³ Freie Universität Berlin, Takustraße 9, D-14195 Berlin, Germany
knauer@inf.fu-berlin.de

⁴ Institute of Information & Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{esther,rene}@cs.uu.nl

⁵ School of Electronics and Information Engineering,
Hankuk University of Foreign Studies, Yongin, Korea
cssin@hufs.ac.kr

Abstract. Let P be a simple polygon. We define a *witness set* W to be a set of points such that if any (prospective) guard set G guards W , then it is guaranteed that G guards P . Not all polygons admit a finite witness set. If a finite minimal witness set exists, then it cannot contain any witness in the interior of P ; all witnesses must lie on the boundary of P , and there can be at most one witness in the interior of every edge. We give an algorithm to compute a minimum witness set for P in $O(n^2 \log n)$ time, if such a set exists, or to report the non-existence within the same time bounds.

1 Introduction

Visibility problems have been studied extensively in the Computational Geometry literature, and the so-called *art gallery problems* form an important subcategory within this field. The problem of how many points or *guards* are always sufficient to guard any polygon with n vertices was posed by Victor Klee in 1976. Chvátal [1] showed soon thereafter that $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient, and sometimes necessary. Since then, a lot of research in this field has been carried out, and in 1987 O'Rourke published a whole book on the topic [2], while many new results that had been achieved after the publication of O'Rourke's book have appeared in surveys by Shermer [3] and Urrutia [4].

Most of the papers related to the Art Gallery Problem research consider the computation of the location of the guards under various restrictions on the shape of the polygon (i.e., orthogonal polygons, polygons with holes, etc.) or on the placement of the guards (i.e., edge guards, vertex guards, mobile guards, etc.).

Except for the case of a single guard, only few papers consider the computation of the guarded region for a ... set of guards [5, 6].

Approximately seven years ago, Joseph Mitchell posed the ... to Tae-Cheon Yang during a research visit of the latter: “Given a polygon P , does it admit a ... , i.e., a set of objects in P such that any (prospective) guard set that guards the witnesses is guaranteed to guard to whole polygon?” The simplest kind of witnesses are points, possibly constrained to lie on the vertices or on the boundary of P . The idea, of course, is that in the case of a set of moving guards, or of a guard set that permits the addition or removal of guards, it is easier to check point-to-point (i.e., guard-to-witness) visibility, than to update the complete visibility region of all guards. Other possible types of witnesses are edges of the polygon. Yang showed this problem to a student, and they made an initial effort to classify polygons that can be witnessed by guards placed at vertices at the polygons, and polygons that can be witnessed by (partial) edges of the polygon; see [7] (in Korean).

In this paper, we consider point witnesses that are allowed to lie anywhere in the interior or on the boundary of the polygon. We want to determine for a given polygon P whether a finite witness set exists, and if this is the case, to compute a ... witness set. The main contribution of our paper is the combinatorial/geometrical result that a finite minimum size witness set for a polygon (if it exists) contains only witnesses on the boundary, and there are at most n of them, where n is the number of vertices of the polygon. Furthermore, we show that for any $n \geq 4$ there is a polygon that can be witnessed by no less than $n - 2$ witnesses. The results are not trivial: there are polygons that are not witnessable by a finite witness set, but that need an infinite number of witnesses on their boundary or in their interior. There are also polygons for which there are witness sets that witness the boundary, but not the whole interior of the polygon. Examples of such polygons can be found in the full version of this paper, and in a preliminary version [8]. Note that in this preliminary version we only give results concerning minimal (but not necessarily minimum size) witness sets.

For completeness, we also outline an algorithm to compute a minimum size witness set for a polygon if such a set exists, or to report the non-existence otherwise. The running time of this algorithm is $O(n^2 \log n)$.

The paper is organized as follows: in the next section we introduce the formal definition of ... , and we prove some interesting basic properties of them. Using the results from Section 2, we study properties of ... witness sets in Section 3. In Section 4, we discuss the cardinality of minimum size witness sets. In Section 5 we give an algorithm for computing minimum size witness sets, and we wrap up in Section 6 with a brief discussion on our results and on open problems.

2 Preliminaries

Throughout this paper, P denotes a simple polygon with n vertices. We assume that P 's vertices $V(P) = \{v_0, v_1, \dots, v_{n-1}\}$ are ordered in counterclockwise direction. We also assume that the vertices of P are in general position,

that is, no three vertices of P are collinear¹. The edges of P are denoted with $E(P) = \{e_0, e_1, \dots, e_{n-1}\}$, with $e_i = (v_i, v_{(i+1) \bmod n})$. Geometrically, we consider the edge e_i to be the closed line segment between its incident vertices, i.e., an edge includes its endpoints. Edges e_i are directed from v_i to v_{i+1} , so that the interior of P lies locally to the left of e_i . We say that a point p lies in P if p lies in the interior of P (denoted with $\text{int}(P)$) or on its boundary (denoted with ∂P), i.e., we consider P to be a closed subset of \mathbb{E}^2 .

A point p in P sees a point q in P if the line segment pq is contained in P . Since polygons are closed regions, the line-of-sight pq is not blocked by grazing contact with the boundary of P ; this definition of visibility is commonly used in the Art Gallery literature [2]. We say that a point p in P sees a reflex vertex v of P if p sees v , and the edges incident to v do not lie on different sides of the line through p and v (i.e., one of the edges may lie on this line).

Let e be an edge e of P . Let $\ell(e)$ be the directed line through e such that $\ell(e)$ has the same orientation as e . The positive halfspace induced by $\ell(e)$ is the region of points in \mathbb{E}^2 to the left of $\ell(e)$, and we denote it by $\ell^+(e)$. The negative halfspace $\ell^-(e)$ is defined analogously. The kernel of a (possibly open) region of points $R \subset \mathbb{E}^2$ is the union of R and its boundary ∂R ; we denote it with $\text{cl}(R)$.

For an edge $e \in E(P)$, $\ell(e) \cap P$ consists of one or more connected parts (segments) that lie completely in P or on its boundary. The segment that contains e is denoted by $s(e)$. Similarly, for a point p in P and a vertex v of P , $\ell(p, v)$ is defined as the line through p and v , and $\ell(p, v) \cap P$ consists of one or more connected parts (segments) that lie completely in P or on its boundary. The segment that contains p and v is denoted by $s(p, v)$.

The kernel of a polygon P is the set of points from which every point in P is visible. If the kernel is nonempty, we call P kernel-visible. It is well-known that the kernel of a polygon P can be described as $\bigcap_{e \in E(P)} \text{cl}(\ell^+(e))$.

Let p be a point in P . The visibility polygon of p is the set of points in P that are visible to p . We denote the visibility polygon by $\text{VP}(p)$. We define the kernel of a point p to be the kernel of its visibility polygon and we denoted it by $\text{VK}(p)$. The visibility polygon $\text{VP}(p)$ of a point p in P is star-shaped by definition (the kernel contains at least p).

Definition 1. A witness set W of a polygon P is a set of points $W \subset P$ such that every point p in P has a line segment gp to a point g in W that does not intersect the boundary of P other than at p . We call the elements of W witnesses. A guard set G of a polygon P is a set of points $G \subset P$ such that every point p in P has a line segment gp to a point g in G that does not intersect the boundary of P other than at p .

The elements of G in the above definition are commonly referred to as guards, and we call the elements of a witness set witnesses. Note that a guard set always is nonempty, but that a witness set is allowed to be empty (namely, the empty set is a witness set for any convex polygon).

The following theorem states the necessary and sufficient conditions on witness sets:

¹ This assumption is only used in the proof of Lemma 17.

Theorem 1. $W \subseteq P \implies W \text{ is a witness set for } P$

Let $W = \{w_1, \dots, w_k\}$ be a set of points in P such that $\cup_{w \in W} \text{VK}(w)$ covers P completely. Let G be an arbitrary set of points in P such that every element $w_i \in W$ is visible from at least one $g_j \in G$. Since g_j lies in $\text{VP}(w_i)$, we have that all points in $\text{VK}(w_i)$ are visible from g_j . Since there is such a g_j for every w_i , and $\cup_{w \in W} \text{VK}(w)$ covers P completely, it follows that every point in P is visible from at least one $g_j \in G$, and therefore W is a witness set for P .

Conversely, let W be an arbitrary witness set for P . Let us assume for the sake of contradiction that $\cup_{w \in W} \text{VK}(w)$ does not cover P completely.

Let us first consider the case where the union of the visibility polygons (as opposed to the visibility kernels) of all elements of W do not cover P completely. In this case, a contradiction is easily derived: place a guard g_i at every witness w_i . Now every witness is seen by at least one guard, but the guards do not see the whole polygon, so W is not a witness set for P .

It remains to consider the more interesting case where the union of the visibility polygons of all elements of W cover P completely. Then we pick an arbitrary point p in the region of P that is not covered by any of the visibility kernels of the witnesses. For any $w_i \in W$, p may or may not lie in $\text{VP}(w_i)$, but in either case, since p does not lie in $\text{VK}(w_i)$, p cannot see any points in $\text{VP}(w_i)$. This means that for each $w_i \in W$ we can place a guard g_i in $\text{VP}(w_i)$ such that g_i does not see p . So every witness w_i is seen by at least one guard (namely, g_i), but the guards do not see every point in P (for none of the guards sees p). This means that W is not a witness set for P , and we have a contradiction again. \square

We also apply the concept of witnesses to individual points. For two points p and q in a polygon P , we say that p is a witness for q (or alternatively, that p witnesses q), if any point that sees p also sees q .

The following lemma is analogous to Theorem 1, and we omit the proof (which is much simpler than the proof of the theorem):

Lemma 1. $p \text{ witnesses } q \implies P \cap \text{VK}(p) \text{ witnesses } q$

The following two lemmas show that witnessing is transitive:

Lemma 2. $P \cap \text{VP}(p) \subseteq \text{VP}(q) \implies P \cap \text{VP}(p) \text{ witnesses } q$

If a point p witnesses a point q , then $q \in \text{VK}(p)$ by the preceding lemma. This implies that everything that is visible from p is also visible from q , which means that $\text{VP}(p) \subseteq \text{VP}(q)$.

Conversely, if $\text{VP}(p) \subseteq \text{VP}(q)$, then any point that sees p , and therefore lies in $\text{VP}(p)$, also lies in $\text{VP}(q)$, and consequently sees q . \square

Lemma 3. $P \cap \text{VP}(p, q) \text{ witnesses } r \implies P \cap \text{VP}(p) \text{ witnesses } r$

If p witnesses q and q witnesses r , then by the preceding lemma, $VP(p) \subseteq VP(q)$ and $VP(q) \subseteq VP(r)$. This means that $VP(p) \subseteq VP(r)$ and thus that p witnesses r . \square

This leads to the notion of **witness set**. A set W is called a **witness set** for P if W is a witness set for P and, for any $w \in W$, $W \setminus \{w\}$ is not a witness set for P . The proofs of the lemmas in the rest of this section are straightforward, and we omit them here. The interested reader can find them in a preliminary version of this paper [8], as well as in the full version.

Lemma 4. Let P be a polygon and W a witness set for P . If $w \in W$, $w' \in W$, $w' \neq w$, then $VP(w) \cap VP(w') \subseteq P$.

Lemma 5. Let P be a polygon and W a witness set for P . If $W' \subseteq W$ and $W'' \supseteq W$, then W' is a witness set for P if and only if W'' is a witness set for P .

Lemma 6. Let P be a polygon and p a point in P . Then $VP(p) \cap P = \{p\}$.

Lemma 7. Let p and q be points in P . If p witnesses q , then $VP(p) \cap VP(q) \subseteq P$.

Lemma 8. Let p be a point in P . If $v \in V(P)$, then $VP(p) \cap VP(v) = \{p\}$.

Lemma 9. Let P be a polygon and p a point in P . If W is a witness set for P , then $W \cap VP(p) = \{p\}$.

By combining Lemmas 4, 7 and 8, we get the following lemma:

Lemma 10. Let P be a polygon and W a witness set for P . If $|W| > 1$, then $W \cap VP(w) = \{w\}$ for any $w \in W$.

3 Finite Witness Sets

In this section we bound the number of witnesses of a finite minimal witness set W for a polygon P from above. We show first that the elements of W can only lie on the boundary of P (Lemma 11), and next, that any edge of P has at most one element of W in its interior (Lemma 13).

We denote the regions of points in P witnessed by any of the elements of a set S of points in P by $\mathcal{R}(S)$. If S is finite, then $\mathcal{R}(S)$ consists of one or more polygonal regions, since it is the union of a finite set of kernels of visibility polygons, which are closed. Note that in degenerate cases, a region in $\mathcal{R}(S)$ may be a single point or a line segment. The regions of points that are not witnessed by any of the elements of S are denoted by $\mathcal{Q}(S) = P \setminus \mathcal{R}(S)$, and

these are called the *regions*. $\mathcal{Q}(S)$ consists of one or more connected (but not necessarily simply connected) polygonal regions that are either open (if the region lies in $int(P)$) or neither open nor closed (if the region contains part of ∂P). The important fact is that no part of $\partial \mathcal{Q}(S)$ that lies in $int(P)$ belongs to $\mathcal{Q}(S)$ itself.

Lemma 11. *If P is a simple polygon and W is a finite witness set for P , then $W \cap int(P) = \emptyset$.*

If W is the empty set, then the lemma holds trivially.

W cannot have only one element. Otherwise, by Lemma 9, P would be convex, and since the empty set is also a valid witness set for convex polygons, this would contradict the minimality of W .

It remains to show that the lemma holds in case W has more than one element. In this case, consider any witness $w \in W$. Note that $\mathcal{Q}(W \setminus \{w\})$ cannot be a point or a one-dimensional region; otherwise, $\mathcal{R}(W \setminus \{w\})$ would not be closed, and this is only possible if W is an infinite set. From Lemma 4 we deduce that w lies in $\mathcal{Q}(W \setminus \{w\})$. Since no part of $\partial \mathcal{Q}(W \setminus \{w\})$ that lies in $int(P)$ belongs to $\mathcal{Q}(W \setminus \{w\})$ itself, w can only lie in $int(\mathcal{Q}(W \setminus \{w\}))$ or on $\mathcal{Q}(W \setminus \{w\}) \cap \partial P$. By Lemma 10, w lies on the boundary of $VK(w)$. Since w cannot lie simultaneously on the boundary of $VK(w)$ (which is closed) and in the open set $int(\mathcal{Q}(W \setminus \{w\}))$, the conclusion is that w can only lie on $\mathcal{Q}(W \setminus \{w\}) \cap \partial P$. □

We established that witnesses of a finite minimal witness set W lie on the boundary of P . Using the following lemma, we show in Lemma 13 that every edge of P has at most one element of W in its interior.

Lemma 12. *If P is a simple polygon and W is a finite witness set for P , then for any edge e of P and any witness $w \in W$, w lies on e if and only if w is an endpoint of e .*

Since W is a finite witness minimal witness set for P , any $w \in W$ lies on some edge e of P by Lemma 11. Note that w lies on two such edges if it lies on a vertex. Now suppose, for the sake of contradiction, that w sees past one or more reflex vertices v of P that do not lie on $\ell(e)$. Then there is at least one reflex vertex v such that w sees past v and $VK(w)$ is contained in $cl(\ell^+(w, v))$.

We now regard only the boundary of P . The intersection of any visibility kernel (which is convex) with ∂P consists of one or more parts that are homeomorphic to a point or a to a closed line segment. Similarly, since W is finite, $\mathcal{R}(W \setminus \{w\}) \cap \partial P$ consists of one or more parts that are homeomorphic to a point or a closed line segment, and $\mathcal{Q}(W \setminus \{w\}) \cap \partial P$ consists of one or more parts that are homeomorphic to an open line segment.

The remainder of the proof is essentially a one-dimensional version of the proof of Lemma 11. Since (i) w lies on the intersection of e and $\ell(w, v)$, (ii) $VK(w)$ is contained in $cl(\ell^+(w, v))$, and (iii) w sees past v , w must be the endpoint of one of the parts of $VK(w) \cap \partial P$. Since W is a minimal witness set, by Lemma 4, w must also lie in $\mathcal{Q}(W \setminus \{w\})$. But w cannot lie simultaneously in an open

subset and on the boundary thereof, so we conclude that w does not see past any reflex vertex that does not lie on $\ell(e)$. \square

Note that the above lemma does not contradict Lemma 7. A witness w on an edge e sees the vertices of P that lie on $\ell(e)$, and combining Lemmas 7 and 12 implies that w must see past at least one of these vertices.

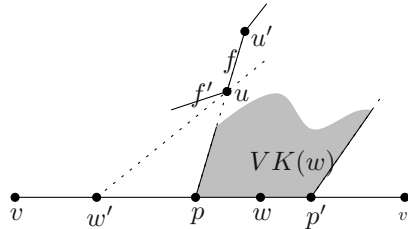


Fig. 1. Illustration to the proof of Lemma 13. Any point w' in between v and p sees past u or (not shown) past another reflex vertex in the triangle formed by w', u , and p

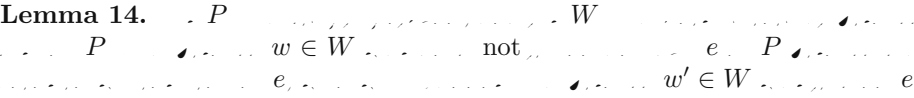
Lemma 13. P W
 P e P $w \in W$
 $w' \in W$ e

Let e be an edge of P that contains a witness $w \in W$ in its interior. Let e' be the closed segment $e \cap VK(w)$, and let p and p' be the endpoints of e' . No other witness $w' \in W$ can lie on e in between p and p' (otherwise, w' would lie in $VK(w)$, and by Lemma 4 this contradicts the minimality of W). Therefore, if p and p' coincide with the vertices v and v' incident to e , then the lemma holds.

So let us consider the situation that not both of p and p' coincide with v and v' , respectively, and let us assume without loss of generality that p does not coincide with v ; see Figure 1. Since p lies on the boundary of $VK(w)$, p must be the intersection of e and $s(f)$ (the extension of an edge f , as defined in Section 2) for some edge f of P , visible from w . Let u and u' be the vertices incident to f , with u the one closest to e . Let f' be the other edge incident to u . Note that f' must also be visible from w , otherwise, w would see past u , and that would contradict Lemma 12. Now, for the sake of contradiction, suppose that there is a witness $w' \in W$ that lies in the half-open segment from (and including) v to (but not including) p . If w' sees u , then w' also sees past u . Otherwise, if w' doesn't see u , there must be a reflex vertex u'' on the shortest geodesic path from w' to u such that w' sees past u'' (note that in the latter case, u'' must lie in the interior of the triangle formed by w', u , and p). In either case we derive a contradiction with Lemma 12, and the conclusion is that the half-open segment from v to p cannot contain any witness. Analogously, if p' doesn't coincide with v' , then the half-open segment from v' to p' cannot contain a witness either. \square

We omit the proof of the following lemma; it can be found in a preliminary version of this paper [8] and in the full version. The main arguments use

Lemma 12. The lemma itself is used to prove Lemma 15, which in turn is applied in Section 4.3, where we investigate the possible locations of the witnesses on the boundary of P .

Lemma 14.  $w \in W$ not $w' \in W$

Lemma 15.  $w \in W$ $w' \in W$

If w does not witness e completely, there must be at least one other witness w' that witnesses a part of e . By Lemma 13, w' cannot lie on e . We know that w' witnesses at least one point on the interior of e , since the part of e that does lie in $VK(w)$ cannot consist of only one or both vertices of e (because visibility kernels are closed regions). But then by Lemma 14, there cannot be a witness that lies on e – and this contradicts the fact that w lies on e . \square

Note that the situation depicted in Figure 1, where the points p and p' lie in between the vertices v and v' , cannot occur if W is a finite witness set.

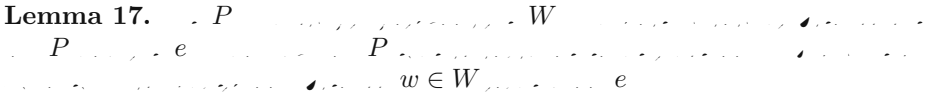
Lemma 16.  $w \in W$ $w' \in W$

Suppose, for the sake of contradiction, that there is an element w of W that lies on a reflex vertex v of P . Observe that $VK(w)$ does not contain any points on the edges e and e' incident to v , except v itself. This means that all points on e and e' (except v) must be witnessed by the elements of $W \setminus \{w\}$. However, the union of the visibility kernels of these witnesses is a closed region, since W is finite, and therefore v is also covered by these visibility kernels. This means that w is witnessed by another witness $w' \in W$, and this contradicts the minimality of W . \square

4 Minimum Size Witness Sets

In this section we classify the edges of P into three categories, depending on whether their incident vertices are convex or not: we distinguish reflex-reflex edges, convex-reflex (or reflex-convex) edges, and convex-convex edges. In Section 4.1 we show that each reflex-reflex and convex-reflex edge of P contains exactly one witness, and in the following section we show that no witness lies on a convex-convex edge, except possibly at its vertices that are also incident to a convex-reflex edge. In Section 4.3 finally, we show to place witnesses on reflex-reflex and convex-reflex edges, namely: for each reflex-reflex edge we can place a witness anywhere in its interior, and for each convex-reflex edge we can place a witness on its convex edge. This placement strategy establishes a minimum size witness set for P —if a finite witness set for P exists. Testing whether the candidate witness set is indeed a witness set is the subject of Section 5.

4.1 Reflex-Reflex and Convex-Reflex Edges



Let v be a reflex vertex of e . There must be a witness w of W that witnesses a point p in the interior of e , as well as v itself, since $VK(w)$ is a closed region, and we have only finitely many witnesses.

We first show that w must lie on $s(e)$ (recall from Section 2 that $s(e)$ is the single piece of $\ell(e)$ that lies in $int(P)$ and that contains e). Suppose, for the sake of contradiction, w does not lie on $s(e)$. Obviously, both v and p are visible from w , so w must lie in $\ell^+(e)$, and by Lemma 11 w lies on the boundary of P . We have two cases to consider:

- w sees past v . However, according to 12, this is only allowed if v lies on $\ell(f)$, where f is the edge on which w lies. However, f cannot be incident to v (otherwise v would not be reflex), so this violates the assumption made in Section 2 that no three vertices of P are collinear.
- w does not see past v . Since w sees v , we have that w sees both edges e and e' incident to v and $VK(w)$ is completely contained in $cl(\ell^+(e)) \cap cl(\ell^+(e'))$. But since v is reflex, this means that the only point on e that is contained in $VK(w)$ is v , contradicting the fact that w also witnesses p .

Since both cases lead to a contradiction, we conclude that w lies on $s(e)$.

Since w lies on the boundary of P by Lemma 11, w lies either on e or on one of the endpoints of $s(e)$. Let q be an endpoint of $s(e)$. q is either a convex vertex of e , or it does not lie on e . In the latter case, q sees a reflex vertex v' of e as well as both edges e and e' incident to v' (v' may or may not be v). Were w located on q in that case, then we argue again that the only point on e that is contained in $VK(w)$ is v' , contradicting the fact that w also witnesses p . We conclude that w lies on e .

By Lemma 16 w does not lie on a reflex vertex of e , and therefore we consider the following two cases:

- w lies in the interior of e . By Lemma 13, no other witness besides w can lie on e ;
- w lies on the convex vertex of e (this case is only applicable if e is a convex-reflex edge). By Lemma 13, no other witness can lie in $int(e)$. By Lemma 16, the remaining reflex vertex of e also cannot contain a witness, so w is the only witness on e .

In both cases, w is the only witness on e and thus the proof is completed. \square

4.2 Convex-Convex Edges

No witness of a minimal witness set is located on a convex-convex edge, except possible at a vertex that is also incident to a convex-reflex edge:

Lemma 18. 

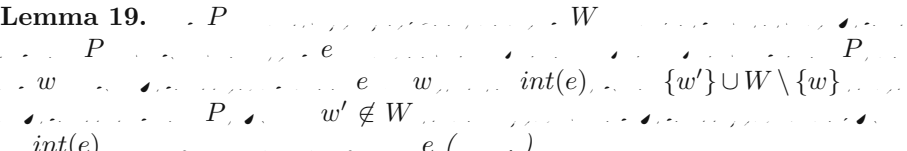
If W is the empty set, then the lemma holds trivially.

W cannot have only one element. Otherwise, by Lemma 9, P would be convex, and since the empty set is also a valid witness set for convex polygons, this would contradict the minimality of W .

It remains to show that the lemma holds in case W has more than one element. In this case, suppose for the sake of contradiction, that there is a witness $w \in W$ that lies either in the interior of a convex-convex edge e , or on a convex vertex that is shared by two convex-convex edges e and e' . There is at least one other witness $w' \in W$, and w lies outside $VK(w')$; otherwise, w' would witness w , and this contradicts the minimality of W . By Lemma 7 w sees past at least one reflex vertex v of P , and by Lemma 12 v lies on $\ell(e)$ (or on $\ell(e')$). However, all vertices incident to e (and e') are convex, so this is impossible. \square

4.3 Placement of Witnesses

By Lemma 17, every finite minimal witness set W for P contains exactly one element w located on a convex-reflex or reflex-reflex edge. The following lemma shows to be useful in constructing a minimum size witness set for a polygon.

Lemma 19. 

If w' and w coincide, the lemma follows trivially. If on the other hand w and w' lie on different points on e , then we argue that w' witnesses w , and therefore, w' can replace w , as by Lemma 3, any point that is witnessed by w is also witnessed by w' .

Suppose, for the sake of contradiction, that w' does not witness w , or in other words, that w does not lie in $VK(w')$. This means that the boundary of $VK(w')$ intersects e in a point p that lies in between w and w' . This point p is the endpoint of some segment $s(f)$, (partially) visible from w' , but not visible from w . Let u be the vertex of f closest to p . We have that u is a reflex vertex, and moreover, u lies in $\ell^+(e)$. Now, if u is visible from w , then w sees past u . Otherwise, if u is not visible from w , there must be a reflex vertex u' on the shortest geodesic path from w to u such that w sees past u' (note that in the latter case, u' must lie in the interior of the triangle formed by w , u , and p). In either case, we derive a contradiction with lemma 12, and the conclusion is that w' indeed witnesses w . \square

Combining the lemmas in Sections 3 and 4 we derive our main theorem:

Theorem 2. For any simple polygon P , there exists a minimum size witness set W for P such that $|W| \leq n - 2$.

Furthermore, we have that for every $n \geq 4$ there is a polygon that is witnessable with no less than $n - 2$ witnesses, as is illustrated in Figure 2.

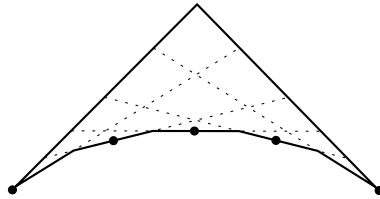


Fig. 2. For any $n \geq 4$ there is a polygon with n vertices that is witnessable with no less than $n - 2$ witnesses

5 Algorithm

In this section, we outline an algorithm that computes a minimum size witness set W for a simple polygon P with n vertices, if such a set exists, or reports the non-existence of such a set otherwise. The idea is straightforward: we place witnesses according to Theorem 2, giving W , and check whether the union of the visibility kernels of the witnesses covers the whole polygon. This can be done by computing the arrangement induced by the visibility kernels and by P itself, using a sweepline algorithm [9]. During the sweep, we maintain for every cell in the arrangement whether it is covered by at least one visibility kernel, or only by P . In the latter case, we report that P is not witnessable by a finite witness set. Otherwise, we report that W is a witnessset for P .

Since each visibility kernel has $O(n)$ edges and we have $O(n)$ witnesses, we have $O(n^2)$ edges in total, and the arrangement has $O(n^4)$ complexity. However, we can show that every visibility kernel has at most two edges that are (part of the) edges of P . This leads to an arrangement of $O(n^2)$ complexity, and gives a running time of $O(n^2 \log n)$ for our algorithm. Details can be found in the full version of this paper, as well as in a preliminary version [8].

6 Concluding Remarks

We showed that if a polygon P admits a finite witness set, then no minimal witness set W for P has any witnesses in the interior of P , and a minimum size witness set for P can be constructed in linear time. If it is not known in advance whether P is witnessable with a finite witness set, then checking

whether the constructed (candidate) witness set witnesses the polygon can be done in $O(n^2 \log n)$ time.

An interesting direction for further research is to consider other types of witnesses, such as (a subset of) the edges of the polygon. We believe that we can extend our current lemmas, theorems, and algorithms to test whether a polygon is witnessable by an minimal infinite witness set, where all the witnesses lie on the boundary of the polygon.

References

1. Chvátal, V.: A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B* **18** (1975) 39–41
2. O’Rourke, J.: *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY (1987)
3. Shermer, T.C.: Recent results in art galleries. *Proc. IEEE* **80** (1992) 1384–1399
4. Urrutia, J.: Art gallery and illumination problems. In Sack, J.R., Urrutia, J., eds.: *Handbook of Computational Geometry*. North-Holland (2000) 973–1027
5. Cheong, O., van Oostrum, R.: The visibility region of points in a simple polygon. In: *Proc. 11th Canad. Conf. Comp. Geom.* (1999) 87–90
6. Gewali, L., Meng, A., Mitchell, J.S.B., Ntafos, S.: Path planning in $0/1/\infty$ weighted regions with applications. *ORSA J. Comput.* **2** (1990) 253–272
7. Yang, T.C., Shin, C.S.: Guard sufficiency set for polygons. *Journal of Korean Information Science and Technology* **28** (2001) 73–79
8. Chwa, K.Y., Jo, B.C., Knauer, C., Moet, E., van Oostrum, R., Shin, C.S.: Guarding art galleries by guarding witnesses. Report UU-CS-2003-044, Institute for Information & Computing Sciences, Utrecht University, Utrecht, Netherlands (2003)
9. Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.* **C-28** (1979) 643–647
10. Joe, B., Simpson, R.B.: Correction to Lee’s visibility polygon algorithm. *BIT* **27** (1987) 458–473
11. Hershberger, J., Suri, S.: A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms* **18** (1995) 403–431
12. Mulmuley, K.: A fast planar partition algorithm, I. *J. Symbolic Comput.* **10** (1990) 253–280

On p -Norm Based Locality Measures of Space-Filling Curves

H. K. Dai and H. C. Su*

Computer Science Department, Oklahoma State University,
Stillwater, Oklahoma 74078, U.S.A.
{dai, suh}@cs.okstate.edu

Abstract. A discrete space-filling curve provides a linear indexing or traversal of a multi-dimensional grid space. We present an analytical study on the locality properties of the 2-dimensional Hilbert curve family. The underlying locality measure, based on the p -normed metric d_p , is the maximum ratio of $d_p(v, u)^m$ to $d_p(\tilde{v}, \tilde{u})$ over all corresponding point-pairs (v, u) and (\tilde{v}, \tilde{u}) in the m -dimensional grid space and (1-dimensional) index space, respectively. Our analytical results close the gaps between the current best lower and upper bounds with exact formulas for $p \in \{1, 2\}$, and extend to all reals $p \geq 2$.

1 Preliminaries

Discrete space-filling curves have many applications in databases, parallel computation, algorithms, in which linearization techniques of multi-dimensional arrays or grids are needed. Sample applications include heuristics for Hamiltonian traversals, multi-dimensional space-filling indexing methods, image compression, and dynamic unstructured mesh partitioning.

For positive integer n , denote $[n] = \{1, 2, \dots, n\}$. An m -dimensional (discrete) space-filling curve of length n^m is a bijective mapping $C : [n^m] \rightarrow [n]^m$, thus providing a linear indexing/traversal or total ordering of the grid points in $[n]^m$. An m -dimensional grid is said to be of order k if it has side-length $n = 2^k$; a space-filling curve has order k if its codomain is a grid of order k . The generation of a sequence of multi-dimensional space-filling curves of successive orders usually follows a recursive framework (on the dimensionality and order), which results in a few classical families, such as Gray-coded curves, Hilbert curves, Peano curves, and z-order curves (see, for examples, [1] and [8]).

Denote by H_k^m and Z_k^m an m -dimensional Hilbert and z-order, respectively, space-filling curve of order k . Figure 1 illustrates the recursive constructions of H_k^2 and Z_k^2 for $m = 2$, and $k = 1, 2$. The locality preservation of a space-filling curve family reflects proximity between the grid points of $[n]^m$, that is, close-by points in $[n]^m$ are mapped to close-by indices/numbers in $[n^m]$, or vice

* Current address: Department of Computer Science, Arkansas State University, State University, Arkansas 72467, U.S.A.

versa. Clustering performance measures the distribution of continuous runs of grid points (clusters) over identically shaped subspaces of $[n]^m$, which can be characterized by the average number of clusters and the average inter-cluster distance (in $[n^m]$) within a subspace.

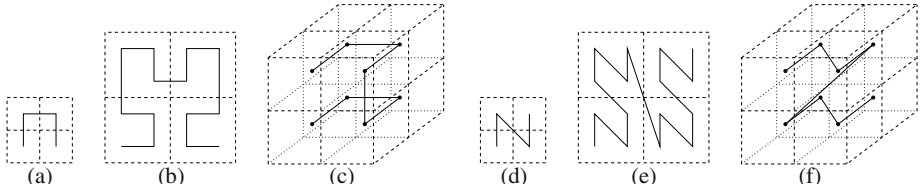


Fig. 1. Recursive constructions of Hilbert and z-order curves of higher order (H_k^m and Z_k^m , respectively) by interconnecting symmetric (via reflection and rotation) subcurves of lower order (H_{k-1}^m and Z_{k-1}^m , respectively) along an order-1 subcurve (H_1^m and Z_1^m , respectively): (a) H_1^2 ; (b) H_2^2 ; (c) H_3^3 ; (d) Z_1^2 ; (e) Z_2^2 ; (f) Z_3^3

Empirical and analytical studies of clustering performances of various low-dimensional space-filling curves have been reported in the literature (see [8], and [3] for details). Generally, the Hilbert and z-order curve families exhibit good performance in this respect. Moon, Jagadish, Faloutsos, and Saltz [8] prove that in a sufficiently large m -dimensional H_k^m -structural grid space, the mean number of clusters over all rectilinear polyhedral queries with surface area $S_{m,k}$ approaches $\frac{1}{2} \frac{S_{m,k}}{m}$ as k approaches ∞ . They also obtain the exact formula for the mean number of clusters over all rectangular $2^q \times 2^q$ subgrids of an H_k^2 -structural grid space.

The space-filling index structure can support efficient query processing (such as range queries) provided that we minimize the average number of external fetch/seek operations, which is related to the clustering statistics. Asano, Ranjan, Roos, Welzl, and Widmayer [2] study the optimization of range queries over space-filling index structures, which aims at minimizing the number of seek operations (not the number of block accesses) — tradeoff between seek time to proper block (cluster) and latency/transfer time for unnecessary blocks (inter-cluster gap). Good bounds on inter-clustering statistics translate into good bounds on the average tolerance of unnecessary block transfers.

Dai and Su [3] obtain the exact formulas for the following three statistics for H_k^2 and Z_k^2 : (1) the summation of all inter-cluster distances over all $2^q \times 2^q$ query subgrids, (2) the universe mean inter-cluster distance over all inter-cluster gaps from all $2^q \times 2^q$ subgrids, and (3) the mean total inter-cluster distance over all $2^q \times 2^q$ subgrids.

The studies above show that the Hilbert curve family manifests good data clustering properties, robust mathematical formalism, and viable indexing techniques for querying multi-dimensional data, when compared with other curve families. This paper presents an analytical study on the locality properties of the 2-dimensional Hilbert curve family. The underlying locality measure, based on the p -normed metric d_p , is the maximum ratio of $d_p(v, u)^m$ to $d_p(\tilde{v}, \tilde{u})$ over

all corresponding point-pairs (v, u) and (\tilde{v}, \tilde{u}) in the m -dimensional grid space and (1-dimensional) index space, respectively. Our analytical results close the gaps between the current best lower and upper bounds with exact formulas for $p \in \{1, 2\}$, and extend to all reals $p \geq 2$. We also verify the results with computer programs over various p -values and grid-orders.

2 Locality Measures and Related Work

To analyze the locality of a space-filling curve family, we need to rigorously define its measures that are practical — good bounds (lower and upper) on the locality measure translate into good bounds on the declustering (locality loss) in one space in the presence of locality in the other space. A few locality measures have been proposed and analyzed for space-filling curves in the literature. Denote by d and d_p the Euclidean metric and p -normed metric (Manhattan ($p = 1$) and maximum metric ($p = \infty$)), respectively.

Let \mathcal{C} denote a family of m -dimensional curves of successive orders. For quantifying the proximity preservation of close-by points in the m -dimensional space $[n]^m$, Pérez, Kamata, and Kawaguchi [9] employ an average locality measure:

$$L_{\text{PKK}}(C) = \sum_{i,j \in [n^m] | i < j} \frac{|i - j|}{d(C(i), C(j))} \text{ for } C \in \mathcal{C},$$

and provide a hierarchical construction for a 2-dimensional \mathcal{C} with good but suboptimal locality with respect to this measure.

Mitchison and Durbin [7] use a more restrictive locality measure parameterized by q :

$$L_{\text{MD},q}(C) = \sum_{i,j \in [n^m] | i < j \text{ and } d(C(i), C(j))=1} |i - j|^q \text{ for } C \in \mathcal{C}$$

to study optimal 2-dimensional mappings for $q \in [0, 1]$. For the case $q = 1$, the optimal mapping with respect to $L_{\text{MD},1}$ is very different from that in [9]. For the case $q < 1$, they prove a lower bound for arbitrary 2-dimensional curve C :

$$L_{\text{MD},q}(C) \geq \frac{1}{1 + 2q} n^{1+2q} + O(n^{2q}),$$

and provide an explicit construction for 2-dimensional \mathcal{C} with good but suboptimal locality. They conjecture that the space-filling curves with optimal locality (with respect to $L_{\text{MD},q}$ with $q < 1$) must exhibit a “fractal” character.

Dai and Su [5] consider a locality measure similar to $L_{\text{MD},1}$ conditional on a 1-normed distance of δ between points in $[n]^m$:

$$L_\delta(C) = \sum_{i,j \in [n^m] | i < j \text{ and } d_1(C(i), C(j))=\delta} |i - j| \text{ for } C \in \mathcal{C}.$$

They derive exact formulas for L_δ for the Hilbert curve family $\{H_k^m \mid k = 1, 2, \dots\}$ and z-order curve family $\{Z_k^m \mid k = 1, 2, \dots\}$ for $m = 2$ and arbitrary δ that is an integral power of 2, and $m = 3$ and $\delta = 1$:

$$\begin{aligned}
 L_\delta(H_k^2) &= \begin{cases} \frac{17}{2 \cdot 7} \cdot 2^{3k} - \frac{5}{2 \cdot 3} \cdot 2^{2k} - \frac{2^3}{3 \cdot 7} & \text{if } \delta = 1 \\ \frac{17}{2 \cdot 7} \cdot 2^{2k+2 \log \delta} - \frac{2^3 \cdot 3 \cdot 5 \cdot 7^{(k-\log \delta)+5 \cdot 7 \cdot 383}}{2^4 \cdot 3^3 \cdot 5 \cdot 7} \cdot 2^{2k+3 \log \delta} \\ + \frac{2 \cdot 3 \cdot 5 \cdot (k-\log \delta) - 1}{2^2 \cdot 3^3} \cdot 2^{2k+\log \delta} - \frac{2^2 \cdot 41}{3 \cdot 3 \cdot 5 \cdot 7} \cdot 2^{5 \log \delta} - \frac{2}{3 \cdot 3} \cdot 2^{3 \log \delta} - \frac{2}{3 \cdot 5} \cdot 2^{\log \delta} & \text{otherwise} \end{cases} \\
 L_\delta(Z_k^2) &= \begin{cases} 2^{3k} - 2^k & \text{if } \delta = 1 \\ 2^{3k+2 \log \delta} - \left(\frac{2}{3^2}(k - \log \delta) + \frac{1949}{2^5 \cdot 3 \cdot 7}\right) 2^{2k+3 \log \delta} \\ + \left(\frac{2}{3^2}(k - \log \delta) + \frac{7}{2^2 \cdot 3^3}\right) 2^{2k+\log \delta} + \frac{19}{2^2 \cdot 3 \cdot 7} \cdot 2^{2k} - \frac{2^2}{7} \cdot 2^{k+4 \log \delta} \\ - \frac{3}{7} \cdot 2^{k+\log \delta} + \frac{2 \cdot 5}{3^3 \cdot 7} \cdot 2^{5 \log \delta} - \frac{2^2}{3^3} \cdot 2^{3 \log \delta} + \frac{2}{3 \cdot 7} \cdot 2^{2 \log \delta} & \text{otherwise} \end{cases} \\
 L_1(H_k^3) &= \frac{67}{2 \cdot 31} \cdot 2^{5k} - \frac{11}{2 \cdot 7} \cdot 2^{3k} - \frac{2^6}{7 \cdot 31} \\
 L_1(Z_k^3) &= 2^{5k} - 2^{2k}
 \end{aligned}$$

With respect to the locality measure L_δ and for sufficiently large k and $\delta \ll 2^k$, the z -order curve family performs better than the Hilbert curve family for $m = 2$ and over the δ -spectrum of integral powers of 2. When $\delta = 2^k$, the domination reverses. The superiority of the z -order curve family persists but declines for $m = 3$ with unit 1-normed distance for L_δ .

For measuring the proximity preservation of close-by points in the indexing space $[n^m]$, Gotsman and Lindenbaum [6] consider the following measures for $C \in C$:

$$L_{GL,\min}(C) = \min_{i,j \in [n^m] | i < j} \frac{d(C(i), C(j))^m}{|i - j|}, \text{ and } L_{GL,\max}(C) = \max_{i,j \in [n^m] | i < j} \frac{d(C(i), C(j))^m}{|i - j|}.$$

They show that for arbitrary m -dimensional curve C ,

$$L_{GL,\min}(C) = O(n^{1-m}), \text{ and } L_{GL,\max}(C) > (2^m - 1)(1 - \frac{1}{n})^m.$$

For the m -dimensional Hilbert curve family $\{H_k^m \mid k = 1, 2, \dots\}$, they prove that:

$$L_{GL,\max}(H_k^m) \leq 2^m(m + 3)^{\frac{m}{2}}.$$

For the 2-dimensional Hilbert curve family, they obtain tight bounds:

$$6(1 - O(2^{-k})) \leq L_{GL,\max}(H_k^2) \leq 6 \frac{2}{3}.$$

Alber and Niedermeier [1] generalize $L_{GL,\max}$ to $L_{AN,p}$ by employing the p -normed metric d_p in place of the Euclidean metric d . They improve and extend the above tight bounds for the 2-dimensional Hilbert curve family to:

$$L_{AN,1}(H_k^2) \leq 9 \frac{3}{5}, \quad 6(1 - O(2^{-k})) \leq L_{AN,2}(H_k^2) \leq 6 \frac{1}{2}, \text{ and } 6(1 - O(2^{-k})) \leq L_{AN,\infty}(H_k^2) \leq 6 \frac{2}{5}.$$

We focus our analytical study on the locality measure $L_{AN,p}$ for the 2-dimensional Hilbert curve family, and obtain exact formulas for $L_{AN,p}(H_k^2)$ for $p = 1$ and all reals $p \geq 2$.

3 Exact Formulas for $L_{AN,p}(H_k^2)$ with $p \geq 2$

For 2-dimensional Hilbert curves, the self-similar structural property guides us to decompose H_k^2 into four identical H_{k-1}^2 -subcurves (via reflection and rotation), which are amalgamated together by an H_1^2 -curve. Following the linear order along this H_1^2 -curve, we denote the four H_{k-1}^2 -subcurves (quadrants) as $Q_1(H_k^2)$, $Q_2(H_k^2)$, $Q_3(H_k^2)$, and $Q_4(H_k^2)$.

We extend the notations to identify all H_l^m -subcurves of a structured H_k^m for all $l \in [k]$ inductively on the order. Let $Q_i(H_k^m)$ denote the i -th H_{k-1}^m -subcurve (along the amalgamating H_1^m -curve) for all $i \in [2^m]$. Then for the i -th H_{l-1}^m -subcurve, $Q_i(H_l^m)$, of H_l^m , where $2 < l \leq k$ and $i \in [2^m]$, let $Q_j(Q_i(H_l^m))$ denote the j -th H_{l-2}^m -subcurve of $Q_i(H_l^m)$ for all $j \in [2^m]$. We write $Q_i^{q+1}(H_l^m)$ for $Q_i(Q_i^q(H_l^m))$ for all $l \in [k]$ and all positive integers $q < l$. The notation $Q_i^l(H_l^m)$ identifies the i -th grid point in the H_1^m -subcurve $Q_i^{l-1}(H_l^m)$.

For a 2-dimensional Hilbert curve H_k^2 indexing the grid $[2^k]^2$, with a canonical orientation shown in Figure 2(a), we denote by $\partial_1(H_k^2)$ and $\partial_2(H_k^2)$ the entry and the exit, respectively, grid points in $[2^k]^2$ (with respect to the canonical orientation). Figure 2 depicts the decomposition of H_k^2 and the ∂_1 - and ∂_2 -labels of four H_{k-1}^2 -subcurves.

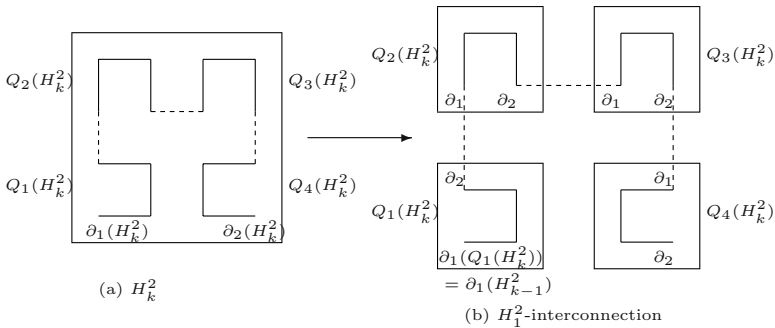


Fig. 2. Generation of H_k^2 from a H_1^2 -interconnection of four H_{k-1}^2 -subcurves

For an H_l^m -subcurve C of a 2-dimensional Hilbert curve H_k^m in Cartesian x - y coordinates, where $l \in [k]$, notice that $\partial_1(C)$ and $\partial_2(C)$ differ in exactly one coordinate, say $z \in \{x, y\}$. We say that the subcurve C is z^+ -oriented (respectively, z^- -oriented) if the z -coordinate of $\partial_1(C)$ is less than (respectively, greater than) that of $\partial_2(C)$. Note that for a 2-dimensional Hilbert curve H_k^m , its two subcurves $Q_2(H_k^m)$ and $Q_3(H_k^m)$ inherit the orientation from their supercurve H_k^2 .

For a space-filling curve C indexing an m -dimensional grid space, the notation “ $v \in C$ ” refers to “grid point v indexed by C ”, and $C^{-1}(v)$ gives the index of v in the 1-dimensional index space. The locality measure in our study is:

$$L_{AN,p}(C) = \max_{\text{indices } i, j \in [n^m]} \frac{d_p(C(i), C(j))^m}{d_p(i, j)} = \max_{v, u \in C} \frac{d_p(v, u)^m}{|C^{-1}(v) - C^{-1}(u)|}.$$

When $m = 2$, we write $\mathcal{L}_{C,p}(v, u) = \frac{d_p(v,u)^2}{\delta_C(v,u)}$, where $\delta_C(v, u)$ denotes the index-difference $|C^{-1}(v) - C^{-1}(u)|$. We generalize the notations $L_{AN,p}(C)$ and $\mathcal{L}_{C,p}$ for a subcurve C (of a space-filling curve) in an obvious manner. A pair of grid points v and u is representative for C with respect to $L_{AN,p}$ if $\mathcal{L}_{C,p}(v, u) = L_{AN,p}(C)$.

In order to obtain exact formulas for $L_{AN,p}(H_k^2)$ for all reals $p \geq 2$, it suffices to consider identifying all representative pairs (v, u) that yield $\mathcal{L}_{H_k^2,2}(v, u) = L_{AN,2}(H_k^2)$, due to the monotonicity of the underlying p -normed metric. A refined analysis based on the upper-bound argument in [6] reveals that the representative pair resides in a subcurve C composed of four linearly adjacent Hilbert subcurves. In the following two sections, we derive the exact formula for $L_{AN,2}(C)$, which is used to deduce that for $L_{AN,2}(H_k^2)$

3.1 Locality of Four Linearly Adjacent Hilbert Subcurves

For a 2-dimensional Hilbert curve H_l^2 with $l \geq 3$, there exists a subcurve C that is composed of four linearly adjacent H_k^2 -subcurves with $k \leq l - 3$. Figure 3 depicts the arrangement in Cartesian coordinates. Denote the leftmost and rightmost (first and fourth in the traversal order) H_k^2 -subcurves by ${}_1H_k^2$ (y^- -oriented) and ${}_4H_k^2$ (y^+ -oriented), respectively.

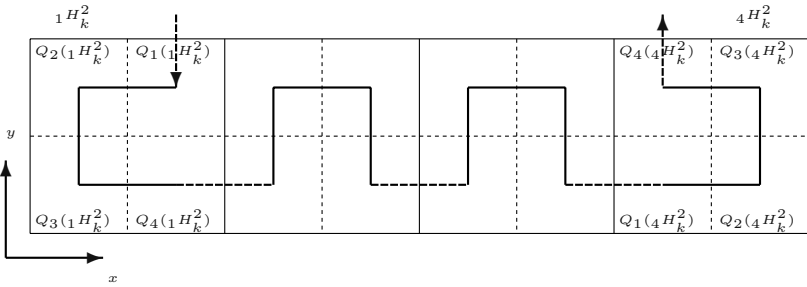


Fig. 3. Four linearly adjacent H_k^2 -subcurves

For a grid point v , denote by $X(v)$ and $Y(v)$ the x - and y -coordinate of v , respectively, and by $(X(v), Y(v))$ the grid point v in the coordinate system. In this subsection, we assume that the lower-left corner grid point of ${}_1H_k^2$ is the origin $(1, 1)$ of the coordinate system. In the following analysis, we identify a pair of grid points $v' \in {}_1H_k^2$ and $u' \in {}_4H_k^2$ such that $\mathcal{L}_{C,2}(v', u') = \max\{\mathcal{L}_{C,2}(v, u) \mid v \in {}_1H_k^2 \text{ and } u \in {}_4H_k^2\}$. Later we see that (v', u') serves as the representative pair for C with respect to $L_{AN,2}$.

To locate a potential representative pair $v \in {}_1H_k^2$ and $u \in {}_4H_k^2$, the following two lemmas and theorem show that the possibility “ $v \in Q_3({}_1H_k^2)$ and $u \in Q_3({}_4H_k^2)$ ” is reduced to seeking v in successive Q_3 -subcurves of ${}_1H_k^2$.

Lemma 1. $v \in Q_3(1H_k^2) - Q_3(Q_3(1H_k^2))$, $u \in Q_3(4H_k^2)$,
 $v' \in Q_3(Q_3(1H_k^2))$, $\mathcal{L}_{C,2}(v', u) \geq \mathcal{L}_{C,2}(v, u)$

Proof. Noting that $Q_3(1H_k^2) - Q_3(Q_3(1H_k^2)) = Q_1(Q_3(1H_k^2)) \cup Q_2(Q_3(1H_k^2)) \cup Q_4(Q_3(1H_k^2))$, we consider the following three cases.

Case 1: $v \in Q_2(Q_3(1H_k^2))$. Consider $v' \in Q_3(Q_3(1H_k^2))$ with $X(v') = X(v)$, then $d_2(v', u)^2 > d_2(v, u)^2$ and $\delta_C(v', u) < \delta_C(v, u)$, and we have $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$.

Case 2: $v \in Q_1(Q_3(1H_k^2))$. Consider $v'' \in Q_2(Q_3(1H_k^2))$ with $Y(v'') = Y(v)$, then $\mathcal{L}_{C,2}(v'', u) \geq \mathcal{L}_{C,2}(v, u)$. From Case 1, there exists $v' \in Q_3(Q_3(1H_k^2))$ such that $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v'', u) > \mathcal{L}_{C,2}(v, u)$.

Case 3: $v \in Q_4(Q_3(1H_k^2))$. Consider $v' \in Q_3(Q_3(1H_k^2))$ with $X(v') = 1$ and $Y(v') = Y(v)$, we have:

$$\begin{aligned} & d_2(v', u)^2 \delta_C(v, u) - d_2(v, u)^2 \delta_C(v', u) \\ &= ((X(u) - X(v'))^2 + (Y(u) - Y(v'))^2)(\delta_C(v, \partial_2(1H_k^2)) + 2 \cdot 2^{2k} + \delta_C(u, \partial_1(4H_k^2)) + 1) \\ &\quad - ((X(u) - X(v))^2 + (Y(u) - Y(v))^2)(\delta_C(v', \partial_2(1H_k^2)) + 2 \cdot 2^{2k} + \delta_C(u, \partial_1(4H_k^2)) + 1) \\ &= ((X(u) - 1)^2)(\delta_C(v, \partial_2(1H_k^2)) + 2 \cdot 2^{2k} + \delta_C(u, \partial_1(4H_k^2)) + 1) \\ &\quad + (Y(u) - Y(v))^2(\delta_C(v, \partial_2(1H_k^2)) + 2 \cdot 2^{2k} + \delta_C(u, \partial_1(4H_k^2)) + 1) \\ &\quad - ((X(u) - X(v))^2)(\delta_C(v', \partial_2(1H_k^2)) + 2 \cdot 2^{2k} + \delta_C(u, \partial_1(4H_k^2)) + 1) \\ &\quad - (Y(u) - Y(v))^2(\delta_C(v', \partial_2(1H_k^2)) + 2 \cdot 2^{2k} + \delta_C(u, \partial_1(4H_k^2)) + 1) \\ &= X(u)^2(\delta_C(v, \partial_2(1H_k^2)) - \delta_C(v', \partial_2(1H_k^2))) + (-2X(u) + 1 + 2X(u)X(v) - X(v)^2)(2 \cdot 2^{2k} + 1) \\ &\quad + (-2X(u) + 1 + 2X(u)X(v) - X(v)^2)(\delta_C(v', \partial_2(1H_k^2)) + \delta_C(u, \partial_1(4H_k^2))) \\ &\quad + (Y(u) - Y(v))^2 \delta_C(v, \partial_2(1H_k^2)) - (Y(u) - Y(v))^2 \delta_C(v', \partial_2(1H_k^2)) \\ &= X(u)^2(\delta_C(v, \partial_2(1H_k^2)) - \delta_C(v', \partial_2(1H_k^2))) + (2X(u) - 1)(X(v) - 1)(2 \cdot 2^{2k} + 1) \\ &\quad + (2X(u) - 1)(X(v) - 1)(\delta_C(v', \partial_2(1H_k^2)) + \delta_C(u, \partial_1(4H_k^2))) \\ &\quad + (Y(u) - Y(v))^2(\delta_C(v, \partial_2(1H_k^2)) - \delta_C(v', \partial_2(1H_k^2))). \end{aligned}$$

Noting that $u \in Q_3(4H_k^2)$, and $v \in Q_4(Q_3(1H_k^2))$ and its corresponding $v' \in Q_3(Q_3(1H_k^2))$, we have:

$$\begin{aligned} 4 \cdot 2^k &\geq X(u) \geq \frac{7}{2} \cdot 2^k + 1, & \frac{1}{2} \cdot 2^k &\geq X(v) \geq \frac{1}{4} \cdot 2^k + 1, \\ 2^k &\geq Y(u) \geq \frac{1}{2} \cdot 2^k + 1, & \frac{1}{4} \cdot 2^k &\geq Y(v) \geq 1; \\ \frac{5}{16} \cdot 2^{2k} &> \delta_C(v, \partial_2(1H_k^2)) &\geq \frac{1}{4} \cdot 2^{2k}, \\ \frac{5}{16} \cdot 2^{2k} &> \delta_C(v', \partial_2(1H_k^2)) &\geq \frac{5}{16} \cdot 2^{2k}. \end{aligned}$$

These bounds yield the following lower bounds for the four terms appearing in $d_2(v', u)^2 \delta_C(v, u) - d_2(v, u)^2 \delta_C(v', u)$:

$$\begin{aligned} & X(u)^2(\delta_C(v, \partial_2(1H_k^2)) - \delta_C(v', \partial_2(1H_k^2))) \geq -2 \cdot 2^{4k}, \\ & (2X(u) - 1)(X(v) - 1)(2 \cdot 2^{2k} + 1) \geq (7 \cdot 2^k + 1)(\frac{1}{4} \cdot 2^k)(2 \cdot 2^{2k} + 1) \geq \frac{7}{2} \cdot 2^{4k}, \\ & (2X(u) - 1)(X(v) - 1)(\delta_C(v', \partial_2(1H_k^2)) + \delta_C(u, \partial_1(4H_k^2))) \geq (7 \cdot 2^k + 1)(\frac{1}{4} \cdot 2^k)(\frac{9}{16} \cdot 2^{2k}) > 0, \\ & (Y(u) - Y(v))^2(\delta_C(v, \partial_2(1H_k^2)) - \delta_C(v', \partial_2(1H_k^2))) \geq (2^k - 1)^2(-\frac{2}{16} \cdot 2^{2k}) > -\frac{1}{8} \cdot 2^{4k}. \end{aligned}$$

These give that $d_2(v', u)^2 \delta_C(v, u) - d_2(v, u)^2 \delta_C(v', u) > 0$, hence $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$.

Combining the three cases, the lemma is proved. \blacksquare

Lemma 2. $h \in \mathbb{N}$, $1 \leq h < k$, $v \in Q_3^h(1H_k^2) - Q_3^{h+1}(1H_k^2)$,
 $u \in Q_3(4H_k^2)$, $v' \in Q_3^{h+1}(1H_k^2)$, $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$

Proof. Similar to the proof of the previous lemma. \blacksquare

Theorem 3. $h \in \mathbb{N}, 1 \leq h < k, v \in Q_3^h(1H_k^2) - Q_3^k(1H_k^2), u \in Q_3(4H_k^2), v' \in Q_3^k(1H_k^2), \mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$

Proof. By induction on $k - h$. For the basis of the induction ($k - h = 1$), apply Lemma 2 with $h = k - 1$.

For the induction step, suppose that the statement in the lemma is true for all integers h with $1 \leq k - h < n$, where $n > 1$. Consider the case when $k - h = n$. Let $v \in Q_3^h(1H_k^2) - Q_3^k(1H_k^2)$ and $u \in Q_3(4H_k^2)$ be arbitrary. Since $Q_3^h(1H_k^2) = Q_3(Q_3^h(1H_k^2)) \cup (Q_1(Q_3^h(1H_k^2)) \cup Q_2(Q_3^h(1H_k^2)) \cup Q_4(Q_3^h(1H_k^2))) = Q_3^{h+1}(1H_k^2) \cup (Q_3^h(1H_k^2) - Q_3^{h+1}(1H_k^2))$, we consider the following two cases.

Case 1: $v \in Q_3^{h+1}(1H_k^2)$. Notice that $k - (h + 1) < n$. Apply the induction hypothesis for the case of $k - (h + 1)$, we obtain a desired v' .

Case 2: $v \in Q_3^h(1H_k^2) - Q_3^{h+1}(1H_k^2)$. By Lemma 2, there exists $v' \in Q_3^{h+1}(1H_k^2)$ such that $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$. If $v' \in Q_3^k(1H_k^2)$, then v' is a desired grid point. Otherwise ($v \in Q_3^{h+1}(1H_k^2) - Q_3^k(1H_k^2)$), this is reduced to Case 1.

This completes the induction step, and the lemma is proved. ■

Theorem 3 says that the lower-left corner grid point v' with coordinates $(1, 1)$ is unique in $Q_3(1H_k^2)$ such that $\mathcal{L}_{C,2}(v', u) = \max\{\mathcal{L}_{C,2}(v, u) \mid v \in Q_3(1H_k^2)\}$ for arbitrary $u \in Q_3(4H_k^2)$.

The search for a potential representative pair can be reduced to a case analysis for all possible pair-combinations $(Q_i(1H_k^2), Q_j(4H_k^2))$ for all $i, j \in \{1, 2, 3, 4\}$. After eliminating symmetrical cases and grouping, it suffices to analyze five major cases: $(Q_3(1H_k^2), Q_2(4H_k^2))$, $(Q_3(1H_k^2), Q_3(4H_k^2))$, $(Q_3(1H_k^2), Q_4(4H_k^2))$, $(Q_4(1H_k^2), 4H_k^2)$, and $(Q_1(1H_k^2) \cup Q_2(1H_k^2), Q_3(4H_k^2) \cup Q_4(4H_k^2))$. We show that the analysis for each pair is reduced to that for the pair $(Q_3(1H_k^2), Q_2(4H_k^2))$ in the following four theorems.

Theorem 4. $v \in Q_3(1H_k^2), u \in Q_3(4H_k^2), v' \in Q_3(1H_k^2), u' \in Q_2(4H_k^2), \mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$

Proof. Consider $v' \in Q_3^k(1H_k^2) (= (1, 1))$ and $u' \in Q_2(4H_k^2)$ with $X(u') = X(u)$ and $Y(u') = 1$. A case analysis for $u \in Q_i(Q_3(4H_k^2))$ with $i \in \{1, 2, 3, 4\}$ shows that $\mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v', u)$. By Theorem 3, $\mathcal{L}_{C,2}(v', u) \geq \mathcal{L}_{C,2}(v, u)$; therefore $\mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$. ■

Theorem 5. $v \in Q_3(1H_k^2), u \in Q_4(4H_k^2), v' \in Q_3(1H_k^2), u' \in Q_2(4H_k^2), \mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$

Proof. Consider $u'' \in Q_3(4H_k^2)$ with $Y(u'') = Y(u)$. Notice that $d_2(v, u'') > d_2(v, u)$ and $\delta_C(v, u'') < \delta_C(v, u)$, we have $\mathcal{L}_{C,2}(v, u'') > \mathcal{L}_{C,2}(v, u)$. By Theorem 4, there exist $v' \in Q_3(1H_k^2)$ and $u' \in Q_2(4H_k^2)$ such that $\mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u'') > \mathcal{L}_{C,2}(v, u)$. ■

Lemma 6. $v \in Q_4(1H_k^2), u \in 4H_k^2 (= Q_1(4H_k^2) \cup Q_2(4H_k^2) \cup Q_3(4H_k^2) \cup Q_4(4H_k^2)), v' \in Q_3(1H_k^2), \mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$

Proof. Consider $v' \in Q_3(1H_k^2)$ with $Y(v') = Y(v)$ and $X(v') = 1$. A case analysis for $u \in Q_i(4H_k^2)$ with $i \in \{1, 2, 3, 4\}$ shows that $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$. ■

Theorem 7. $\dots v \in Q_4(1H_k^2) \dots u \in {}_4H_k^2 \dots v' \in Q_3(1H_k^2)$
 $\dots u' \in Q_2(4H_k^2) \dots \mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$

Proof. Lemma 6 says that there exists $v' \in Q_3(1H_k^2)$ such that $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$. Since $u \in {}_4H_k^2 = Q_1(4H_k^2) \cup Q_2(4H_k^2) \cup Q_3(4H_k^2) \cup Q_4(4H_k^2)$, consider four pair-combinations for (v', u) : $(Q_3(1H_k^2), Q_i(4H_k^2))$ with $i \in \{1, 2, 3, 4\}$. The analysis for the pair $(Q_3(1H_k^2), Q_1(4H_k^2))$ is equivalent to that for the pair $(Q_4(1H_k^2), Q_2(4H_k^2))$, which is reduced to $(Q_3(1H_k^2), Q_2(4H_k^2))$ by Lemma 6. The pair $(Q_3(1H_k^2), Q_3(4H_k^2))$ is reduced to $(Q_3(1H_k^2), Q_2(4H_k^2))$ by Theorem 4, and the pair $(Q_3(1H_k^2), Q_4(4H_k^2))$ is reduced to $(Q_3(1H_k^2), Q_2(4H_k^2))$ by Theorem 5. ■

Theorem 8. $\dots v \in Q_1(1H_k^2) \cup Q_2(1H_k^2) \dots u \in Q_3(4H_k^2) \cup Q_4(4H_k^2)$
 $\dots v' \in Q_3(1H_k^2) \dots u' \in Q_2(4H_k^2) \dots \mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$

Proof. Consider $v'' \in Q_3(1H_k^2) \cup Q_4(1H_k^2)$ with $X(v'') = X(v)$ and $Y(v'') = Y(v) - 2^{k-1}$, and $u'' \in Q_1(4H_k^2) \cup Q_2(4H_k^2)$ with $X(u'') = X(u)$ and $Y(u'') = Y(u) - 2^{k-1}$. Since $d_2(v'', u'') = d_2(v, u)$ and $\delta_C(v'', u'') < \delta_C(v, u)$, we have $\mathcal{L}_{C,2}(v'', u'') > \mathcal{L}_{C,2}(v, u)$. It suffices to consider two pair-combinations for (v'', u'') : $(Q_3(1H_k^2), Q_1(4H_k^2))$ and $(Q_4(1H_k^2), Q_1(4H_k^2) \cup Q_2(4H_k^2))$. The analysis for the pair $(Q_3(1H_k^2), Q_1(4H_k^2))$ is equivalent to that for $(Q_4(1H_k^2), Q_2(4H_k^2))$, which is reduced to $(Q_3(1H_k^2), Q_2(4H_k^2))$ by Lemma 6. The pair $(Q_4(1H_k^2), Q_1(4H_k^2) \cup Q_2(4H_k^2))$ is a subcase of Theorem 7. Consequently, for these two pair-combinations for (v'', u'') , there exist $v' \in Q_3(1H_k^2)$ and $u' \in Q_2(4H_k^2)$ such that $\mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v'', u'') > \mathcal{L}_{C,2}(v, u)$, as desired. ■

An immediate consequence of the previous four theorems is summarized below — a representative pair must reside in $(Q_3(1H_k^2), Q_2(4H_k^2))$.

Theorem 9. $\dots v \in {}_1H_k^2 - Q_3(1H_k^2) \dots u \in {}_4H_k^2 - Q_2(4H_k^2)$
 $\dots v' \in Q_3(1H_k^2) \dots u' \in Q_2(4H_k^2) \dots \mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$

The following lemma and theorem complement Lemma 2 and Theorem 3, respectively, with similar proofs. Having reached the pair $(Q_3(1H_k^2), Q_2(4H_k^2))$ for seeking a potential representative pair (v', u') , they guide the search into successive Q_3 -subcurves of ${}_1H_k^2$ for v' . The symmetry in the pair $(Q_3(1H_k^2), Q_2(4H_k^2))$ leads the search into successive Q_2 -subcurves of ${}_4H_k^2$ for u' .

Lemma 10. $\dots h \dots 1 \leq h < k \dots v \in Q_3^h(1H_k^2) - Q_3^{h+1}(1H_k^2)$
 $\dots u \in Q_2(4H_k^2) \dots v' \in Q_3^{h+1}(1H_k^2) \dots$
 $\mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$

Theorem 11. $\dots h \dots 1 \leq h < k \dots v \in Q_3^h(1H_k^2) - Q_3^k(1H_k^2)$
 $\dots u \in Q_2(4H_k^2) \dots v' \in Q_3^k(1H_k^2) \dots \mathcal{L}_{C,2}(v', u) > \mathcal{L}_{C,2}(v, u)$

The following theorem summarizes our analysis above, and asserts that the unique representative pair reside at the lower-left and lower-right corners of C .

Theorem 12. $\dots v \in {}_1H_k^2 - Q_3^k(1H_k^2) \dots u \in {}_4H_k^2 - Q_2^k(4H_k^2)$
 $\dots v' \in Q_3^k(1H_k^2) \dots u' \in Q_2^k(4H_k^2) \dots \mathcal{L}_{C,2}(v', u') > \mathcal{L}_{C,2}(v, u)$
 $\mathcal{L}_{C,2}(v', u') = 6 \cdot \frac{2^{2k+3} - 2^{k+2} + 2^{-1}}{2^{2k+3} + 1}$

Proof. By Theorems 9 and 11 (and its symmetry), we have $v' \in Q_3^k(1H_k^2)$ with coordinates $(1, 1)$ and $u' \in Q_2^k(H_k^2)$ with coordinates $(2^{k+2}, 1)$, which maximizes the $\mathcal{L}_{C,2}$ -value.

Notice that $\delta_C(v', u') = 2(\sum_{i=0}^{k-1} 2^{2i} + 1 + 2 \cdot 2^{2k}) - 1$, hence $\mathcal{L}_{C,2}(v', u') = \frac{d_2(v', u')^2}{\delta_C(v', u')} = 6 \cdot \frac{2^{2k+3} - 2^{k+2} + 2^{-1}}{2^{2k+3} + 1}$. ■

3.2 Exact Formula for $L_{AN,2}(H_k^2)$

The current best bounds for the 2-dimensional Hilbert curve family with respect to $L_{AN,2}(H_k^2)$ [1] is: $6(1 - O(2^{-k})) \leq L_{AN,2}(H_k^2) \leq 6\frac{1}{2}$. Following the argument in [6] with a refined analysis, together with the exact formula for $\mathcal{L}_{C,2}(v', u')$ in Section 3.1, we close the gaps between the two bounds with an exact formula for $L_{AN,2}(H_k^2)$.

Theorem 13. For all $k \geq k_0$, where $k_0 = \dots$, $k \geq k_0$.

$$L_{AN,2}(H_k^2) = 6 \cdot \frac{2^{2k-3} - 2^{k-1} + 2^{-1}}{2^{2k-3} + 1}.$$

Proof. In the upper-bound argument in [6], an arbitrary subcurve/subpath P of length l along H_k^2 is considered. Note that for arbitrary l , there exists a sufficiently large positive integer r such that $(2^{r-1})^2 < l \leq (2^r)^2$. This gives that P is contained in two adjacent quadrants Q' and Q'' , each with size $(2^r)^2$ (grid points). Let D denote the diameter (Euclidean) of the set of grid points in P . A case analysis of subpath containment (of P) in subquadrants of size $(2^{r-1})^2$ within $Q' \cup Q''$ results in the following six cases:

1. $\frac{4}{16} 4^r < l \leq \frac{5}{16} 4^r$: $D^2 < \frac{5}{4} 4^r$, hence $\frac{D^2}{l} \leq 5$.
2. $\frac{5}{16} 4^r < l \leq \frac{6}{16} 4^r$: $D^2 < \frac{29}{16} 4^r$, hence $\frac{D^2}{l} \leq 5\frac{4}{5}$.
3. $\frac{6}{16} 4^r < l \leq \frac{7}{16} 4^r$: $D^2 < \frac{19}{4} 4^r$, hence $\frac{D^2}{l} \leq 6\frac{2}{3}$.
4. $\frac{7}{16} 4^r < l \leq \frac{8}{16} 4^r$: $D^2 < \frac{19}{4} 4^r$, hence $\frac{D^2}{l} \leq 5\frac{5}{7}$.
5. $\frac{8}{16} 4^r < l \leq \frac{12}{16} 4^r$: $D^2 < \frac{13}{4} 4^r$, hence $\frac{D^2}{l} \leq 6\frac{1}{2}$.
6. $\frac{12}{16} 4^r < l \leq 4^r$: $D^2 < 5 \times 4^r$, hence $\frac{D^2}{l} \leq 6\frac{3}{2}$.

In order to obtain the desired $L_{AN,2}$ -bound, it suffices to refine the analysis of subpath containment in Cases 3, 5, and 6 in subquadrants of size $(2^{r-2})^2$. The refined analysis for Case 3 yields the upper bounds on $\frac{D^2}{l}$: $\frac{29}{6}$, $\frac{137}{25}$, $\frac{141}{26}$, and $\frac{160}{27}$ (maximum is $\frac{160}{27} < 5.93$). For Case 6, the upper bounds on $\frac{D^2}{l}$ are: $\frac{68}{12}$, $\frac{73}{13}$, $\frac{80}{14}$, and $\frac{80}{15}$ (maximum is $\frac{80}{14} < 5.72$).

The analysis for Case 5 reveals that all but one arrangement (of subquadrants of size $(2^{r-2})^2$) yield upper bounds that are bounded above and away from 6. The exception structure is given by the subcurve C (described in Section 3.1) of four linearly adjacent Hilbert subcurves H_{k-3}^2 (of order $k - 3$). By Theorem 12, the maximum $\frac{D^2}{l}$ -value for this case is $6 \cdot \frac{2^{2(k-3)+3} - 2^{(k-3)+2} + 2^{-1}}{2^{2(k-3)+3} + 1} (= 6 \cdot \frac{2^{2k-3} - 2^{k-1} + 2^{-1}}{2^{2k-3} + 1})$. Observe that $\frac{2^{2k-3} - 2^{k-1} + 2^{-1}}{2^{2k-3} + 1}$ is strictly increasing (in k) and approaching 1 (as $k \rightarrow \infty$). This establishes the theorem. ■

For an x^+ -oriented Hilbert curve H_k^2 with $\partial_1(H_k^2) = (1, 1)$, where $k \geq k_0$, the representative pair for H_k^2 with respect to $L_{AN,2}$ reside at the lower-left corner (with coordinates $(2^{k-2} + 1, 2^{k-1} + 1)$) and the lower-right corner (with coordinates $(2^k - 2^{k-2}, 2^{k-1} + 1)$) of four linearly adjacent largest subquadrants (H_{k-3}^2 -subcurves).

3.3 Exact Formulas for $L_{AN,p}(H_k^2)$ with $p > 2$

In order to study $L_{AN,p}$ for arbitrary real p , we first investigate the monotonicity of the underlying p -normed metric.

Lemma 14. *Let $f : (0, +\infty) \rightarrow (1, +\infty)$ be defined by $f(p) = (1 + \alpha^p)^{\frac{1}{p}}$, $\alpha > 1$. Then f is strictly decreasing over its domain.*

Proof. It is equivalent to show that the function $g : (0, +\infty) \rightarrow (0, +\infty)$ defined by $g(p) = \log f(p)$ (“log” denotes the natural logarithm) is strictly decreasing over its domain. We consider the first derivative of g , which is defined on $(0, +\infty)$:

$$g'(p) = \frac{\frac{\alpha^p}{1+\alpha^p} \log \alpha^p - \log(1 + \alpha^p)}{p^2} = \frac{\log \alpha^p - \log(1 + \alpha^p) - \frac{\log \alpha^p}{1+\alpha^p}}{p^2}.$$

Clearly, $g'(p) < 0$ for $0 < \alpha < 1$ (from the first equality above), and $g'(p) < 0$ for $1 \leq \alpha$ (from the second equality above). This proves the strictly decreasing property of f over its domain. ■

An immediate corollary of the previous lemma is that for all grid points v and u , the p -normed metric $d_p(v, u)$ is decreasing in $p \in (0, +\infty)$. Hence for a space-filling curve C , $\mathcal{L}_{C,p}(v, u) = \frac{d_p(v,u)^2}{\delta_C(v,u)}$ is decreasing in $p \in (0, +\infty)$, as $\delta_C(v, u)$ is independent of p .

Theorem 15. *Let k_0 be the smallest integer such that $k_0 \geq k_0$.*

$$L_{AN,p}(H_k^2) = 6 \cdot \frac{2^{2k-3} - 2^{k-1} + 2^{-1}}{2^{2k-3} + 1}, \text{ for all reals } p \geq 2.$$

Proof. Let (v', u') be the representative pair for H_k^2 with respect to $L_{AN,2}$, with their coordinates $v' = (2^{k-2} + 1, 2^{k-1} + 1)$ and $u' = (2^k - 2^{k-2}, 2^{k-1} + 1)$. Consider an arbitrary real $p \geq 2$. We show that (v', u') also serves as the unique representative pair for H_k^2 with respect to $L_{AN,p}$, that is, $\mathcal{L}_{H_k^2,p}(v', u') > \mathcal{L}_{H_k^2,p}(v, u)$ with $(v, u) \neq (v', u')$.

Observe that $Y(v') = Y(u')$, which implies that $d_p(v', u') = d_2(v', u')$. Then for arbitrary grid points $v, u \in H_k^2$ with $(v', u') \neq (v, u)$, we have:

$$\begin{aligned} \mathcal{L}_{H_k^2,p}(v', u') &= \frac{d_p(v', u')^2}{\delta_{H_k^2}(v', u')} = \frac{d_2(v', u')^2}{\delta_{H_k^2}(v', u')} = \mathcal{L}_{H_k^2,2}(v', u') \\ &> \mathcal{L}_{H_k^2,2}(v, u) \quad (\text{as } (v', u') \text{ is a representative pair with respect to } \mathcal{L}_{H_k^2,2}) \\ &\geq \mathcal{L}_{H_k^2,p}(v, u) \quad (\text{by the monotonicity of } \mathcal{L}_{H_k^2,p}). \quad \blacksquare \end{aligned}$$

3.4 Exact Formula for $L_{AN,1}(H_k^2)$

Following an argument similar to the one in Sections 3.1 and 3.2 to establish $L_{AN,2}(H_k^2)$, we obtain the exact formula for $L_{AN,1}(H_k^2)$.

Theorem 16. *Let k_0 be the smallest integer such that $k_0 \geq k_0$.*

$$L_{AN,1}(H_k^2) = 9 - 3 \cdot 2^{-k+3} + 2^{-2k+4}.$$

There are two (symmetrical) representative pairs for H_k^2 with respect to $L_{\text{AN},1}$; namely (v', u') and (v'', u'') . For an x^+ -oriented Hilbert curve H_k^2 with $\partial_1(H_k^2) = (1, 1)$, where $k \geq k_0$, the coordinates of (v', u') and (v'', u'') are $((2^{k-1}, 1), (1, 2^k))$ and $((2^{k-1} + 1, 1), (2^k, 2^k))$, respectively. Thus $d_1(v', u') = 2^k + 2^{k-1} - 2$ and $\delta_{H_k^2}(v', u') = 2^{2k-2}$, and $L_{\text{AN},1}(H_k^2) = \mathcal{L}_{H_k^2,1}(v', u') = 9 - 3 \cdot 2^{-k+3} + 2^{-2k+4}$.

4 Conclusion

Our analytical study of the locality properties of the Hilbert curve family, $\{H_k^2 \mid k = 1, 2, \dots\}$, is based on the locality measure $L_{\text{AN},p}$, which is the maximum ratio of $d_p(v, u)^m$ to $d_p(\tilde{v}, \tilde{u})$ over all corresponding point-pairs (v, u) and (\tilde{v}, \tilde{u}) in the m -dimensional grid space and index space, respectively. Our results close the gaps between the current best lower and upper bounds with exact formulas for $p \in \{1, 2\}$, and extend to all reals $p \geq 2$. In addition, we identify all the representative pairs (which realize $L_{\text{AN},p}(H_k^2)$) for $p = 1$ and all reals $p \geq 2$. We also verify the results with computer programs over various p -values ($p \in \{1, 2, 3\}$) and grid-orders ($k \in \{4, 5, \dots, 10\}$). For all reals $p \in [1, 2]$ with sufficiently small granularity, our empirical study in [4] reveals two major sources of representative pairs (v, u) that give $\mathcal{L}_{H_k^2,p}(v, u) = L_{\text{AN},p}(H_k^2)$. A practical implication of our results on $L_{\text{AN},p}(H_k^2)$ is that the exact formulas provide good bounds on measuring the loss in data locality in the index space, while spatial correlation exists in the 2-dimensional grid space.

References

1. J. Alber and R. Niedermeier. On multi-dimensional curves with Hilbert property. *Theory of Computing Systems*, 33(4):295–312, 2000.
2. T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science*, 181(1):3–15, 1997.
3. H. K. Dai and H. C. Su. Approximation and analytical studies of inter-clustering performances of space-filling curves. In *Proceedings of the International Conference on Discrete Random Walks (Discrete Mathematics and Theoretical Computer Science, Volume AC (2003))*, pages 53–68, September 2003.
4. H. K. Dai and H. C. Su. An empirical study of p -norm based locality measures of space-filling curves. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1434–1440. Computer Science Research, Education, and Applications Press, June 2003.
5. H. K. Dai and H. C. Su. On the locality properties of space-filling curves. In T. Ibaraki, N. Katoh, and H. Ono, editors, *Lecture Notes in Computer Science (2906): Algorithms and Computation: 14th International Symposium, ISAAC 2003 Proceedings*, pages 385–394, Springer-Verlag, Berlin Heidelberg, 2003.
6. C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5(5):794–797, 1996.

7. G. Mitchison and R. Durbin. Optimal numberings of an $N \times N$ array. *SIAM Journal on Algebraic and Discrete Methods*, 7(4):571–582, 1986.
8. B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.
9. A. Pérez, S. Kamata, and E. Kawaguchi. Peano scanning of arbitrary size images. In *Proceedings of the International Conference on Pattern Recognition*, pages 565–568. IEEE Computer Society, 1992.

Composability of Infinite-State Activity Automata*

Zhe Dang¹, Oscar H. Ibarra^{2,**}, and Jianwen Su²

¹School of Electrical Engineering and Computer Science,
Washington State University,

Pullman, WA 99164, USA

²Department of Computer Science,

University of California,

Santa Barbara, CA 93106, USA

ibarra@cs.ucsb.edu

Abstract. Let \mathcal{M} be a class of (possibly nondeterministic) language acceptors with a one-way input tape. A system $(A; A_1, \dots, A_r)$ of automata in \mathcal{M} , is *composable* if for every string $w = a_1 \dots a_n$ of symbols accepted by A , there is an assignment of each symbol in w to one of the A_i 's such that if w_i is the subsequence assigned to A_i , then w_i is accepted by A_i . For a nonnegative integer k , a k -lookahead delegator for $(A; A_1, \dots, A_r)$ is a *deterministic* machine D in \mathcal{M} which, knowing (a) the current states of A, A_1, \dots, A_r and the accessible "local" information of each machine (e.g., the top of the stack if each machine is a pushdown automaton, whether a counter is zero or nonzero if each machine is a multicounter automaton, etc.), and (b) the k lookahead symbols to the right of the current input symbol being processed, can uniquely determine the A_i to assign the current symbol. Moreover, every string w accepted by A is also accepted by D , i.e., the subsequence of string w delegated by D to each A_i is accepted by A_i . Thus, k -lookahead delegation is a stronger requirement than composability, since the delegator D must be deterministic. A system that is composable may not have a k -delegator for any k . We look at the decidability of composability and existence of k -delegators for various classes of machines \mathcal{M} . Our results have applications to automated composition of e-services. When e-services are modeled by automata whose alphabet represents a set of activities or tasks to be performed (namely, activity automata), automated design is the problem of "delegating" activities of the composite e-service to existing e-services so that each word accepted by the composite e-service can be accepted by those e-services collectively with each accepting a subsequence of the word, under possibly some Presburger constraints on the numbers and types of activities that can be delegated to the different e-services. Our results generalize earlier ones (and resolve some open questions) concerning composability of deterministic finite automata as e-services to finite automata that are augmented with unbounded storage (e.g., counters and pushdown stacks) and finite automata with discrete clocks (i.e., discrete timed automata).

* The research of Oscar H. Ibarra and Jianwen Su was supported in part by NSF Grants IIS-0101134 and CCR02-08595.

** Corresponding author.

1 Introduction

In traditional automata theory, an automaton is a language acceptor that is equipped with finite memory and possibly other unbounded storage devices such as a counter, a stack, a queue, etc. The automaton “scans” a given input word in a one-way/two-way and nondeterministic/deterministic manner while performing state transitions. As one of the most fundamental concept in theoretical computer science, automata are also widely used in many other areas of computer science, in particular, in modeling and analyzing distributed and concurrent systems. For instance, one may view a symbol a in an input word that is read by the automaton as an input/output signal (event). This view naturally leads to automata-based formal models like I/O automata [18]. On the other hand, when one views symbol a as an (observable) activity that a system performs, the automaton can be used to specify the (observable) behavior model of the system; i.e., an activity automaton of the system. For instance, activity automata have been used in defining an event-based formal model of workflow [23]. Recently, activity (finite) automata are used in [4] to model e-services, which are an emerging paradigm for discovery, flexible interoperation, and dynamic composition of distributed and heterogeneous processes on the web or the Internet. An important goal as well as an unsolved challenging problem in service oriented computing [19] such as e-services is *automated composition*: how to construct an “implementation” of a desired e-service in terms of existing e-services.

To approach the automated composition problem, the technique adopted in [4] has two inputs. One input is a finite set of activity finite automata, each of which models an “atomic” e-service. The second is a desired global behavior, also specified as an activity finite automaton, that describes the possible sequences of activities of the e-service to be composed. The output of the technique is a (deterministic) *delegator* that will coordinate the activities of those atomic e-services through a form of delegation. Finding a delegator, if it exists, was shown to be in EXPTIME. The framework was extended in [12] by allowing “lookahead” of the delegator, i.e., to have the knowledge of future incoming activities. A procedure was given which computes a sufficient amount of lookahead needed to perform delegation; however, the procedure is not guaranteed to terminate.

The models studied in [4, 12] have significant limitations: only regular activities are considered since the underlying activity models are finite automata. In reality, more complex and non-regular activity sequences are possible. For instance, activity sequences describing a session of activities `releaseAs`, `allocateAs`, `releaseBs` and `allocateBs` satisfying the condition that the absolute difference between the number of `releaseAs` and the number of `allocateAs`, as well as the absolute difference between the number of `releaseBs` and the number of `allocateBs`, is bounded by 10 (the condition can be understood as some sort of fairness) are obviously non-regular (not even context-free). Therefore, in this paper, we will use the composition model of [12] but focus on, instead of finite automata, infinite-state (activity) automata. Additionally, automata-theoretic techniques are used in our presentation, which are different from the techniques used in [4, 12]. Notice that the problem is not limited only to e-services. In fact, similar automated design problems were also studied in the workflow context [24, 17] and verification communities (e.g., [5, 1, 21, 16]). In the future, we will also look at how our techniques and results can be applied to these latter problems.

In this paper, we use A_1, \dots, A_r to denote r activity automata (not necessary finite-state), which specify the activity behaviors of some r existing e-services. We use A to denote an activity automaton (again, not necessary finite-state), which specifies the desired activity behavior of the e-service to be composed from the existing e-services.

The first issue concerns *composability*. The system $(A; A_1, \dots, A_r)$ is *composable* if for every string (or sequence) $w = a_1 \dots a_n$ of activities accepted by A , there is an assignment (or delegation) of each symbol in w to one of the A_i 's such that if w_i is the subsequence assigned to A_i , then w_i is accepted by A_i . The device that performs the composition is nondeterministic, in general. We start our discussion with A, A_1, \dots, A_r being restricted counter-machines (finite automata augmented with counters, each of which can be incremented/decremented by 1 and can be tested against 0). One of the restrictions we consider is when the counters are reversal-bounded [14]; i.e., for each counter, the number of alternations between nondecreasing mode and nonincreasing mode is bounded by a given constant, independent of the computation. As an example, the above mentioned release-allocate sequences can be accepted by a deterministic reversal-bounded counter-machine with 4 reversal-bounded counters. We use notations like DFAs or NFAs (deterministic or nondeterministic finite automata) and DCMs or NCMs (deterministic or nondeterministic reversal-bounded counter-machines). In [12], it was shown that composability is decidable for a system $(A; A_1, \dots, A_r)$ of DFAs. We generalize this result to the case when A is an NPCM (nondeterministic pushdown automaton with reversal-bounded counters) and the A_i 's are DFAs. In contrast, we show that it is undecidable to determine, given DFAs A and A_1 and a DCM A_2 with only one 1-reversal counter (i.e., once the counter decrements it can no longer increment), whether $(A; A_1, A_2)$ is composable. We also look at other situations where composability is decidable. Further, we propose alternative definitions of composition (e.g., T-composability) and investigate decidability with respect to these new definitions.

When a system is composable, a composer exists but, in general, it is nondeterministic. The second issue we study concerns the existence of a deterministic delegator (i.e., a deterministic composer) within some resource bound. We adopt the notion of k -lookahead delegator (or simply k -delegator) from [12] but for infinite-state automata. (We note that [4] only studied 0-lookahead delegators.) This special form of a delegator is assumed to be efficient, since in its implementation, the delegator does not need to look back to its delegation history to decide where the current activity shall be delegated. For a nonnegative integer k , a k -delegator for $(A; A_1, \dots, A_r)$ is a DCM D which, knowing (1) the current states of A, A_1, \dots, A_r and the signs of their counters (i.e., zero or non-zero), and (2) the k lookahead symbols (i.e., the k "future" activities) to the right of the current input symbol being processed, can deterministically determine the A_i to assign the current symbol. Moreover, every string w accepted by A is also accepted by D , i.e., the subsequence of string w delegated by D to each A_i is accepted by A_i . Clearly, if a system $(A; A_1, \dots, A_r)$ has a k -delegator for some k , then it must be composable. However, the converse is not true – a system may be composable but it may not have a k -delegator for any k .

In [4], the decidability of the existence of a 0-lookahead delegator (i.e., no lookahead) when the automata (i.e., A, A_1, \dots, A_r) are DFAs was shown to be in EXPTIME. The concept of lookahead was introduced in [12] where the focus was still on DFAs;

algorithms for deciding composability and determining an approximate upper bound on k (if it exists) were obtained. A question left open in [12] is whether there is a decision procedure for determining for a given k , whether a system of DFAs has a k -lookahead delegator. We answer this question positively in this paper, even for the more general case when the automata are not necessarily finite-state (e.g., DCMs). Specifically, we show that it is decidable to determine, given a system $(A; A_1, \dots, A_r)$ of DCMs and a nonnegative integer k , whether the system has a k -lookahead delegator.

Our results generalize to composition and lookahead delegation when we impose some linear constraints on the assignments/delegations of symbols. Doing this allows us to further specify some fairness linear constraint on a delegator. For instance, suppose that we impose a linear relationship, specified by a Presburger relation P , on the numbers and types of symbols that can be assigned to A_1, \dots, A_r . We show that it is decidable to determine for a given k , whether a system $(A; A_1, \dots, A_r)$ of DCMs has a k -delegator under constraint P . However, it is undecidable to determine, given a system $(A; A_1, A_2)$, whether it is composable under constraint P , even when A, A_1, A_2 are DFAs and P involves only the symbols assigned to A_2 .

Composability and existence of k -lookahead delegators for systems consisting of other types of automata can also be defined and we study them as well. In particular, we show that composability is decidable for discrete timed automata [2].

The paper has four sections, in addition to this section. Section 2 defines (actually generalizes) the notion of composability of activity automata and proves that it is undecidable for systems $(A; A_1, A_2)$, where A, A_1 are DFAs and A_2 is a DCM with one 1-reversal counter. It is also undecidable when A, A_1, A_2 are DFAs when a Presburger constraint is imposed on the numbers and types of symbols that can be delegated to A_1 and A_2 . In contrast, composability is decidable for systems $(A; A_1, \dots, A_r)$ when A_1, \dots, A_r are DFAs (even NFAs) and A is an NPCM. Decidability holds for other restricted classes of automata as well. Section 3 introduces T -composability and shows that T -composability is decidable for various automata. Section 4 looks at the decidability of the existence for a given k of a k lookahead delegator and shows, in particular, that it is decidable to determine, given a system $(A; A_1, \dots, A_r)$ of NCMs and a nonnegative integer k , whether the system has a k -delegator (even when A is an NPCM). The decidability holds, even if the delegation is under a Presburger constraint. Section 5 investigates composability of discrete timed automata. Because of space limitation, no proofs are given in this extended abstract. They will be presented in a forthcoming paper.

2 Composability

Recall that, throughout this paper, we will use the following notations: a DFA (NFA) is a deterministic (nondeterministic) finite automaton; DCM (NCM) is a DFA (NFA) augmented with reversal-bounded counters; NPCM (DPCM) is a nondeterministic (deterministic) pushdown automaton augmented with reversal-bounded counters.

Machines with reversal-bounded counters have nice decidable properties (see, e.g., [14, 15, 10]), and the languages they accept have the so-called semilinear property. They

have been useful in showing that various verification problems concerning infinite-state systems are decidable [7, 6, 8, 11, 9, 20].

Assumption: For ease in exposition, we will assume that when we are investigating the composability and k -delegability of a system $(A; A_1, \dots, A_r)$ that the machines operate in real-time (i.e., they process a new input symbol at every step). The results can be generalized to machines with a one-way input tape with a right input end marker, where the input head need not move right at every step, and acceptance is when the machine eventually enters an accepting state at the right end marker. This more general model can accept fairly complex languages. For example, the language consisting of all binary strings where the number of 0's is the same as the number of 1's can be accepted by a DCM which, when given a binary input, uses two counters: one to count the 0's and the other to count the 1's. When the input head reaches the right end marker, the counters are simultaneously decremented, and the machine accepts if the two counters reach zero at the same time. Note that the DCM has two 1-reversal counters. In the constructions in proofs of the theorems, we will freely use these non-real-time models with the input end marker. It is known that nondeterministic such machines have decidable emptiness and disjointness problems but undecidable equivalence problem; however, the deterministic varieties have a decidable containment and equivalence problems [14].

Definition 1. Let $(A; A_1, \dots, A_r)$ be a system of activity automata that are DCMs over input (or activity) alphabet Σ . Assume that each DCM starts in its initial state with its counters initially zero. We say that a word (or a sequence of activities) $w = a_1a_2\dots a_n$ is composable if there is an assignment of each symbol a_i to one of the A_1, \dots, A_r such that if the subsequence of symbols assigned to A_i is w_i , then w_i is accepted by A_i (for $1 \leq i \leq r$). We say that the system $(A; A_1, \dots, A_r)$ is composable if every word w accepted by A is composable.

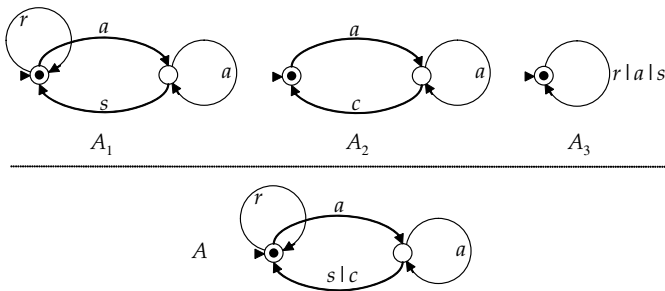


Fig. 1. Four e-Services

Example 1. Consider an online club that offers its customers to first register (represented by r), and then pay for their accesses (a) with either cash (s) or credit cards (c). The e-Service is shown as A in Figure 1, which accepts the language $(r|(aa^*(s|c)))^*$. Assume that there are three existing e-Services, A_1, A_2 , and A_3 , where A_1 handles registration, cash payments for one or more accesses, A_3 is similar to A_1 except that some customers

may use promotion for free accesses, and A_2 can also handle accesses and make credit card transactions. Clearly, the system $(A; A_1, A_2)$ is composable where processing of accesses will be done by whoever collects the payment, cash (A_1) or credit card (A_2).

The system $(A; A_2, A_3)$ is also composable, but in this case, the delegator need only know if the customer will make a credit card payment in the next activity; if so A_2 will perform a , otherwise A_3 does it. Thus this system has a 1-lookahead delegator (to be defined more precisely later). ■

It is known that it is decidable whether a system $(A; A_1, \dots, A_r)$ of DFAs is composable [12]. Somewhat unexpectedly, the following result says that it becomes undecidable when one of the A_i 's is augmented with one 1-reversal counter.

Theorem 1. *It is undecidable to determine, given a system $(A; A_1, A_2)$, where A and A_1 are DFAs and A_2 is a DCM with one 1-reversal counter, whether it is composable.*

Remark 1. Obviously, if the machines are NCMs, composability is undecidable. In fact, take A to be the trivial machine that accepts Σ^* (the universe). Take A_1 to be an arbitrary NCM with one 1-reversal counter. Then the system $(A; A_1)$ is composable iff Σ^* is contained in $L(A_1)$. But the latter problem is known to be undecidable [3]. However, unlike NCMs, equivalence of DCMs is decidable.

Theorem 2. *If A is an NPCM and A_1, \dots, A_r are DFAs (or even NFAs), then composability of $(A; A_1, \dots, A_r)$ is decidable.*

It is of interest to determine the complexity of the composability problem. For example, a careful analysis of the proof of the above theorem and the use of Savitch's theorem that a nondeterministic $S(n)$ space-bounded TM can be converted to an equivalent deterministic $S^2(n)$ space-bounded TM [22], we can show the following:

Corollary 1. *Composability of a system $(A; A_1, \dots, A_r)$ of NFAs can be decided in deterministic exponential space (in the sum of the sizes of the machines).*

There are other cases when composability becomes decidable, if we apply more restrictions to A, A_1, \dots, A_r . A language L is bounded if $L \subseteq w_1^* \dots w_k^*$ for some given k and strings w_1, \dots, w_k (which may not be distinct).

Theorem 3. *Composability is decidable for a system $(A; A_1, \dots, A_r)$ of NCMs when A accepts a bounded language. The result holds even if A and one of the A_i 's are NPCMs.*

Another restriction on the A_i 's is the following. We assume that Σ_i is the input alphabet of A_i . An input symbol a is shared if $a \in \Sigma_i \cap \Sigma_j$ for some $i \neq j$. We say that $(A; A_1, \dots, A_r)$ is n -composable if every word w accepted by A and containing at most n appearances of shared symbols is composable. Then we have:

Theorem 4. *The n -composability of $(A; A_1, \dots, A_r)$ is decidable when A is an NPCM and each A_i is a DCM.*

For our next result, we recall the definitions of semilinear set and Presburger relation [13]. A set $R \subseteq \mathbb{N}^n$ is a *linear set* if there exist vectors v_0, v_1, \dots, v_t in \mathbb{N}^n such that $R = \{v \mid v = v_0 + a_1v_1 + \dots + a_tv_t, a_i \in \mathbb{N}\}$. The vectors v_0 (referred to as the *constant vector*) and v_1, v_2, \dots, v_t (referred to as the *periods*) are called the *generators* of the linear set R . A set $R \subseteq \mathbb{N}^n$ is *semilinear* if it is a finite union of linear sets. It is known that R is a semilinear set if and only if it is a Presburger relation (i.e., can be specified by a Presburger formula).

Let $\Sigma = \{a_1, a_2, \dots, a_n\}$ be an alphabet. For each string w in Σ^* , define the *Parikh map* of w to be $\psi(w) = (num_{a_1}(w), \dots, num_{a_n}(w))$, where $num_{a_i}(x)$ is the number of occurrences of a_i in w . For a language $L \subseteq \Sigma^*$, the *Parikh map* of L is $\psi(L) = \{\psi(w) \mid w \in L\}$.

Let A, A_1, \dots, A_r is a system of DFAs over input alphabet Σ , and P be a Presburger relation (semilinear set). Suppose that we want to check whether the system is composable under constraint P on the numbers and types of symbols that are assigned/delegated to the A_i 's. The constraint is useful in specifying a fairness constraint over the delegations (e.g., it is never true that the absolute value of the difference between the number of activities a assigned to A_1 and the number of activities a assigned to A_2 is larger than 10). Let $\Sigma = \{a_1, \dots, a_n\}$ and P be a Presburger relation (formula) over $(r+1)n$ nonnegative integer variables (note that n is the cardinality of Σ and $r+1$ is the number of the DFAs, including A). The P -composability problem might take the the following form:

Presburger-Constrained Composability Problem: Given a system $(A; A_1, \dots, A_r)$ of DFAs, is the system composable subject to the constraint that for every string $w \in L(A)$, there is an assignment of the symbols in w such that if w_1, \dots, w_r are the subsequences assigned to A_1, \dots, A_r , respectively,

- (1) A_i accepts w_i ($1 \leq i \leq r$).
- (2) $(\psi(w), \psi(w_1), \dots, \psi(w_r))$ satisfies the Presburger relation P .

Unfortunately, because of Theorem 1, the above problem is undecidable:

Corollary 2. *The Presburger-constrained composability problem is undecidable for systems $(A; A_1, A_2)$ of DFAs and a Presburger formula P (even if the formula only involves symbols assigned to A_2).*

3 T-Composability

From the above results, it seems difficult to obtain decidable composability for a system $(A; A_1, \dots, A_r)$ when one or more of A_1, \dots, A_r are beyond DFAs. Below, we will apply more restrictions on how A_1, \dots, A_r are going to be composed such that a decidable composability can be obtained. We define a mapping $T : \Sigma \rightarrow 2^{\{1, \dots, r\}}$ such that each symbol $a \in \Sigma$ is associated with a type $T(a) \subseteq \{1, \dots, r\}$. For $a \in \Sigma$ and $1 \leq i \leq r$, let $(a)_i = a$ if $i \in T(a)$ and $(a)_i = \epsilon$ (the null string) if $i \notin T(a)$. For a string $w = a_1 \dots a_n$, we use $(w)_i$ to denote the result of $(a_1)_i \dots (a_n)_i$. For each A_i , its input alphabet Σ_i consists of all a 's with $i \in T(a)$. Therefore, $(w)_i$ is the result of projecting

w under the alphabet of A_i . We now modify the definition of composability as follows. $(A; A_1, \dots, A_r)$ is T -composable if, for every string w accepted by A , each $(w)_i$ is accepted by A_i . Notice that this definition is different from the original one in the sense that every symbol a in w is assigned to *each* A_i with $i \in T(a)$. Therefore, assignments of symbols in w is deterministic in the new definition (there is a unique way to assign every symbol). One can show:

Theorem 5. *The T -composability of $(A; A_1, \dots, A_r)$ is decidable in the following cases:*

- A is an NPCM and each A_i is a DCM;
- A is an NCM and each A_i is a DPCM.

Theorem 5 does not generalize to the case when one of the A_i 's is an NCM, for the same reason as we stated in Remark 1.

We may take another view of the composition of A_1, \dots, A_r . As we have mentioned earlier, each activity automaton A_i is understood as the behavior specification of an e-service. Each sequence w_i of activities accepted by A_i is an allowable behavior of the service. In the original definition of composability, the activity automata A_1, \dots, A_r are composed through interleavings between the activities in the sequences w_1, \dots, w_r . Clearly, if activities between two services are disjoint, the original definition of composability becomes T -composability with $T(a)$ being a singleton set for every symbol a (i.e., each activity a belongs to a unique activity automaton). When the activity automata share some common activities (e.g., a belongs to both A_1 and A_2 ; i.e., $T(a) = \{1, 2\}$), the T -composability definition implies that an a -activity in A_1 must be synchronized with an a -activity in A_2 . This is why in T -composability, such a symbol a must be assigned to both A_1 and A_2 . Notice that the assignments of each symbol (activity) is deterministic in T -composability. The determinism helps us generalize the above theorem as follows.

A *reset-NCM* M is an NCM that is equipped with a number of *reset states* and is further augmented with a number of *reset counters* (in addition to the reversal-bounded counters). The reset counters are all reset to 0 whenever M enters a reset state. (As usual, we assume that initially the counters start with 0, i.e., with a reset state) We further require that on any execution, the reset counters are reversal-bounded between any two resets. One may similarly define a *reset-NPCM*. Notice that an NCM (resp. NPCM) is a special case of a reset-NCM (resp. reset-NPCM) where there is no reset counter.

Theorem 6. *The emptiness problem for reset-NCMs is decidable.*

We use reset-NPM to denote a reset-NPCM that contains only reset counters and a stack. One can show that the emptiness of reset-NPMs is undecidable.

Theorem 7. *The emptiness problem for reset-NPMs and hence reset-NPCMs is undecidable.*

Now, we generalize Theorem 5 as follows.

Theorem 8. *T -composability of $(A; A_1, \dots, A_r)$ is decidable when A is an NCM and each A_i is a reset-DCM.*

Let NPDA (DPDA) denote a nondeterministic (deterministic) pushdown automaton. Thus, an NPDA is a special case of a reset-NPM, one that does not have reset counters. Using Theorem 7, one can show,

Theorem 9. *T-composability of $(A; A_1, \dots, A_r)$ is undecidable when A is a DPDA and each A_i is a reset-DCM, even for the case when $r = 1$.*

4 Lookahead Delegator

Given k , a k -lookahead delegator (or simply k -delegator) for the system of NCMs $(A; A_1, \dots, A_r)$ is a DCM D which, knowing the current states of A, A_1, \dots, A_r and the statuses (i.e., signs) of their counters (i.e., zero or non-zero), and the k lookahead symbols to the right of the current input symbol being processed, D can uniquely determine the transition of A , the assignment of the current symbol to one of A_1, \dots, A_r , and the transition of the assigned machine. Moreover, for every string x accepted by A , D also accepts, i.e., the subsequence of string x delegated by D to each A_i is accepted by A_i . Clearly, if a system has a k -delegator (for some k), then it must be composable. However, the converse is not true, in general. For example, the system in Figure 1(a) is composable, but it does not have a k -delegator for any k .

Example 2. Consider again Example 1 and in particular the system $(A; A_1, A_2)$. It is easy to see that all a activities immediately preceding an s or c has to be delegated to A_1 or A_2 , respectively. Without knowing which letter, s or c , will be coming, the delegator cannot correctly determine whether A_1 or A_2 should perform the activities a . Thus, the system has no k -delegator for any k . On the other hand, the system $(A; A_2, A_3)$ has a 1-delegator. It is straightforward to generalize this example (by adding additional states) to show that for every k , there exists a system that has a $(k + 1)$ -delegator but not a k -delegator. ■

So that we can always have k lookahead, let $\$$ be a new symbol and f be a new state. Extend the transition function of A by defining the transition from any state, including f , on symbol $\$$ to f . Then make f the only (unique) accepting state. Thus the new NCM accepts the language $L(A)\$+$ and it has only one accepting state f . We can do the same thing for A_1, \dots, A_r with f_1, \dots, f_r their unique accepting states. For convenience, call the new machines also A, A_1, \dots, A_r .

For ease in exposition, in what follows, we assume that $r = 2$, and each of A, A_1, A_2 has only one reversal-bounded counter. Generalizations to any $r \geq 2$ and machines having multiple reversal-bounded counters is straightforward. Note that the transition of A has the form: $\delta_A(q, a, s) = \{\dots, (p, d), \dots\}$, which means that if A is in state q and the input is a and the sign of its counter is s (zero or non-zero), then A can change state to p and increment the counter by d where $d = 0, +1, -1$, with the constraint that if $s = 0$, then $d = 0, +1$. The same holds for transitions δ_1 and δ_2 of A_1 and A_2 . We assume that the counters are initially zero.

Let k be a nonnegative integer. We can construct a candidate k -delegator DCM D as follows: each state of D is a tuple (q, p_1, p_2, u) , where q is a state of A , p_i is a state of A_i , and u is a string of length k . However, in the case (q^0, p_1^0, p_2^0, u) , where q_0 is the initial

state of A and p_i^0 the initial state of A_i , the length of u can be less than k , including zero length, in which case $u = \epsilon$. Then the initial state of D is $(q^0, p_1^0, p_2^0, \epsilon)$. The transition δ of D is defined as follows:

1. $\delta((q^0, p_1^0, p_2^0, \epsilon), 0, 0, 0, a) = ((q^0, p_1^0, p_2^0, a), 0, 0, 0)$ for all symbol a .
2. $\delta((q^0, p_1^0, p_2^0, v), 0, 0, 0, a) = ((q^0, p_1^0, p_2^0, va), 0, 0, 0)$ for all string v such that $|v| < k$ and symbol a .
3. $\delta((q, p_1, p_2, av), s, s_1, s_2, b) = ((q', p'_1, p'_2, vb), d, d_1, d_2)$ for all q, p_1, p_2, s, s_1, s_2 , all string v such that $|v| = k$ and symbols a, b , where:
 - (a) $(q', d) \in \delta_A(q, a, s)$;
 - (b) either $p'_1 = p_1, d_1 = 0$, and $(p'_2, d_2) \in \delta_2(p_2, a, s_2)$
or $p'_2 = p_2, d_2 = 0$, and $(p'_1, d_1) \in \delta_1(p_1, a, s_1)$.

Moreover, the choice $((q', p'_1, p'_2), d, d_1, d_2)$ once made is unique for the parameters $((q, p_1, p_2, av), s, s_1, s_2)$. (Note that, in general, there are many choices that can be made for the given parameters.)

4. Note that in (q, p_1, p_2, u) , any suffix of u may be a string of $\$$'s.
5. Then $(f, f_1, f_2, \$^k)$ is the accepting state of D , where f, f_1, f_2 are the unique accepting states of A, A_1, A_2 .

Now D is a DCM. Since the class of languages accepted by DCMs is effectively closed under complementation, we can construct a DCM E accepting the complement of $L(D)$. Then D is a k -delegator of $(A; A_1, A_2)$ iff $L(A) \cap L(E) = \emptyset$. We can construct from NCM A and DCM E an NCM F accepting $L(A) \cap L(E)$. We can then check the emptiness of $L(F)$ since the emptiness problem for NCMs is decidable. Now D is just one candidate for a k -delegator. There are finitely many such candidates. Every choice that can be made in item 3) above corresponds to one such candidate. By exhaustively checking all candidates, we either find a desired k -delegator or determine that no such k -delegator exists. Thus, we have shown the following:

Theorem 10. *It is decidable to determine, given a system of NCMs $(A; A_1, \dots, A_r)$ and a nonnegative integer k , whether the system has a k -delegator.*

Since the emptiness problem for NPCMs is also decidable, we can generalize the above result to:

Corollary 3. *It is decidable to determine, given a system $(A; A_1, \dots, A_r)$, where A is an NPCM and A_1, \dots, A_r are NCMs, and a nonnegative integer k , whether the system has a k -delegator.*

Corollary 4. *If we impose some Presburger constraint P on the delegation of symbols by the k -delegator (e.g., some linear relationships on the number of symbols delegated to A_1, \dots, A_r), then the existence of such a P -constrained k -delegator is also decidable.*

Open Question: Is it decidable to determine, given a system of DCMs (A, A_1, \dots, A_r) , whether it has a k -delegator for some k ?

Corollary 5. *It is decidable to determine, given a system $(A; A_1, \dots, A_r)$ and a nonnegative integer k , where A is a DPDA (deterministic pushdown automaton), A_1 is a PDA (nondeterministic pushdown automaton) and A_2, \dots, A_r are NFAs, whether the system has a DPDA k -delegator. (Here, the delegation depends also on the top of the stack of A_1 .)*

For the special case when the machines are NFAs, we can prove the following (from the proof of Theorem 10 and Savitch's theorem):

Corollary 6. *We can decide, given a system $(A; A_1, \dots, A_r)$ of NFAs and a nonnegative integer k , whether the system has a k -delegator in nondeterministic exponential time (in k and the sum of the sizes of the machines) and hence, also, in deterministic exponential space.*

5 Composability of Timed Automata

In this section, we study composability of discrete timed automata (DTA) A , which are NFAs augmented with discrete-valued clocks [2]. We say that a word w is *accepted* by A when w is provided on the input tape, if A is able to enter a designated accepting state. We use $L(A)$ to denote the set of words accepted by A . For DTAs, one may develop a similar definition of composability as in Section 2. However, the definition does not justify the intended meaning of composability. For instance, let A_1 and A_2 be two DTAs, and suppose ac (resp. bd) are accepted by A_1 (resp. A_2). Observe that an interleaving like $abcd$ of the two words is not necessarily accepted by the DTA composed from A_1 and A_2 . This is because, when composing, A_1 and A_2 share the same global clock. To devise a proper definition of composability for DTAs, we introduce timed words [2]. A timed word is a sequence of pairs

$$(a_1, t_1) \dots (a_n, t_n) \tag{1}$$

such that each $a_i \in \Sigma$, $t_i \in \mathbf{N}^+$, and $t_1 \leq \dots \leq t_n$. We say that the timed word is *accepted* by A if $w = a_1 \dots a_n$ is accepted by A and this fact is witnessed by some accepting run of A such that each t_i is the timestamp (the value of the global clock) when symbol a_i is read in the run. Thus, the timed word not only records the sequence of symbols $a_1 \dots a_n$ accepted by A but also remembers the timestamp when each symbol is read. Let A, A_1, \dots, A_r be DTAs. A timed word in the form of (1) is *timed composable* if there is an assignment of each pair (a_j, t_j) to one of the A_1, \dots, A_r such that, for $1 \leq i \leq r$, the subsequence (also a timed word) of pairs assigned to A_i is accepted by A_i . We say that $(A; A_1, \dots, A_r)$ is *timed composable* if every timed word accepted by A is timed composable. The main result of this section is the following:

Theorem 11. *The timed composability of discrete timed automata $(A; A_1, \dots, A_r)$ is decidable.*

References

1. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. 16th Int. Colloq. on Automata, Languages and Programming*, 1989.

2. R. Alur and D. Dill. Automata for modeling real-time systems. *Theoretical Computer Science*, 126(2):183–236, 1994.
3. B. Baker and R. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Sciences*, 8:315–332, 1974.
4. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC)*, volume 2910 of *LNCS*, pages 43–58, 2003.
5. J. Buchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
6. Z. Dang. Pushdown time automata: a binary reachability characterization and safety verification. *Theoretical Computer Science*, 302:93–121, 2003.
7. Z. Dang, O. Ibarra, T. Bultan, R. Kemmerer, and J. Su. Binary reachability analysis of discrete pushdown timed automata. In *Proc. Int. Conf. on Computer-Aided Verification (CAV)*, pages 69–84, 2000.
8. Z. Dang, O. H. Ibarra, and R. A. Kemmerer. Generalized discrete timed automata: decidable approximations for safety verification. *Theoretical Computer Science*, 296:59–74, 2003.
9. Z. Dang, O. H. Ibarra, and P. San Pietro. Liveness Verification of Reversal-bounded Multicounter Machines with a Free Counter. In *FSTTCS'01*, volume 2245 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2001.
10. Z. Dang, O. H. Ibarra, and Z. Sun. On the emptiness problems for two-way nondeterministic finite automata with one reversal-bounded counter. In *ISAAC'02*, volume 2518 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2002.
11. Z. Dang, P. San Pietro, and R. A. Kemmerer. Presburger liveness verification for discrete timed automata. *Theoretical Computer Science*, 299:413–438, 2003.
12. C. E. Gerede, R. Hull, and J. Su. Automated composition of e-services with lookahead. Technical report, UCSB, 2004.
13. S. Ginsburg and E. Spanier. Semigroups, presburger formulas, and languages. *Pacific J. of Mathematics*, 16:285–296, 1966.
14. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.
15. O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. New decidability results concerning two-way counter machines. *SIAM J. Comput.*, 24:123–137, 1995.
16. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. IEEE Symposium on Logic In Computer Science*, 2001.
17. S. Lu. *Semantic Correctness of Transactions and Workflows*. PhD thesis, SUNY at Stony Brook, 2002.
18. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. Principles of Distributed Computing*, pages 137–151, 1987.
19. M. Papazoglou. Agent-oriented technology in support of e-business. *Communications of the ACM*, 44(4):71–77, 2001.
20. P. San Pietro and Z. Dang. Automatic verification of multi-queue discrete timed automata. In *COCOON'03*, volume 2697 of *Lecture Notes in Computer Science*, pages 159–171. Springer, 2003.
21. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. IEEE Symp. on Foundations of Computer Science*, 1990.
22. W. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.
23. M. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *Proc. Workshop on Database Programming Languages (DBPL)*, 1995.
24. W. M. P. van der Aalst. On the automatic generation of workflow processes based on product structures. *Computer in Industry*, 39(2):97–111, 1999.

Error Compensation in Leaf Root Problems^{*}

Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13,
D-72076 Tübingen, Fed. Rep. of Germany
{dom, guo, hueffner, niedermr}@informatik.uni-tuebingen.de

Abstract. The k -LEAF ROOT problem is a particular case of graph power problems. Here, we study “error correction” versions of k -LEAF ROOT—that is, for instance, adding or deleting at most l edges to generate a graph that has a k -leaf root. We provide several NP-completeness results in this context, and we show that the NP-complete CLOSEST 3-LEAF ROOT problem (the error correction version of 3-LEAF ROOT) is fixed-parameter tractable with respect to the number of edge modifications in the given graph. Thus, we provide the seemingly first nontrivial positive algorithmic results in the field of error compensation for leaf root problems with $k > 2$. To this end, as a result of independent interest, we develop a forbidden subgraph characterization of graphs with 3-leaf roots.

1 Introduction

Graph powers are a classical concept in graph theory (cf. [2–Section 10.6]) with recently increased interest from an algorithmic point of view. The k -power of a graph $G = (V, E)$ is the graph $G^k = (V, E')$ with $(u, v) \in E'$ iff there is a path of length at most k between u and v in G . We say G is the k -root of G^k ; deciding whether a graph is a power of some other graph is called the k -root problem. It is NP-complete in general [18], but one can decide in $O(|V|^3)$ time whether a graph is a k -power of a tree for any fixed k [12]. In particular, it can be decided in linear time whether a graph is a square of a tree [17, 14]. Very recently, Lau [14] shows that it can be found in polynomial time whether a graph is a square of a bipartite graph, but it is NP-complete to decide whether a graph is a cube of a bipartite graph. Moreover, Lau and Corneil [15] give a polynomial-time algorithm for recognizing k -powers of proper interval graphs for every k and show that, contrariwise, recognizing squares of chordal graphs and split graphs is NP-complete. Here, we concentrate on certain variants of tree powers. Whereas Kearney and Corneil [12] study the problem where every tree node one-to-one corresponds to a graph vertex, Nishimura, Ragde, and Thilikos [21] introduced the notion of k -leaf root where exclusively the tree

^{*} Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

leaves stand in one-to-one correspondence to the graph vertices. Motivated by applications in computational biology, Lin, Kearney, and Jiang [16] and Chen, Jiang, and Lin [4] examine the variant of leaf powers where all inner nodes of the root tree have degree at least three. The corresponding algorithmic problems to decide whether a graph has a k -root are called k -LEAF ROOT [21] and k -PHYLOGENETIC ROOT [16], respectively. For $k \leq 4$, both problems are solvable in polynomial time [21, 16]. The complexities of both problems for $k \geq 5$ are still open. Moreover, Chen et al. [4] show that, under the assumption that the maximum degree of the phylogenetic root is bounded from above by a constant, there is a linear-time algorithm that determines whether a graph has a k -phylogeny for arbitrary k .

What to do if the given input graph has no k -leaf root? In particular, the input graph might be “close” to having a root but due to certain “errors” (as occur in many practical applications), the graph structure would need some “correction” first before the computation of a k -leaf root is doable. This problem was already recognized by Kearney and Corneil [12], and they introduced the CLOSEST k -TREE POWER problem. In this “error correction setting” the question is whether a given graph can be modified by adding or deleting at most l edges such that the resulting graph has a k -tree root. Unfortunately, this problem turns out to be NP-complete for $k \geq 2$ [12, 10]. One also obtains NP-completeness for the corresponding problems CLOSEST k -PHYLOGENETIC ROOT [4] and, as we point out here, CLOSEST k -LEAF ROOT. In addition, for CLOSEST k -LEAF ROOT we study other edge modification problems—namely, only to allow edge deletions or only to allow edge insertions—and we show NP-completeness. See Table 1 in Section 4 for an overview concerning complexity results for CLOSEST k -LEAF ROOT and its variants.

To the best of our knowledge, the above error correction scenario so far only led to results showing hardness of complexity. We are not aware of any results concerning approximation or non-trivial exact algorithms. In contrast, we show the seemingly first positive algorithmic results in this context, proving fixed-parameter tractability with respect to the number l of edge modifications for CLOSEST 3-LEAF ROOT and all its variants mentioned above. To achieve our fixed-parameter results, we develop a novel forbidden subgraph characterization of graphs that are 3-leaf powers—a result that may be of interest on its own: A graph is a 3-leaf power iff it is chordal and it contains none of the 5-vertex graphs bull, dart, and gem as induced subgraph (see Section 3 for details). A much simpler characterization of graphs that are 2-leaf powers is already known by forbidding an induced path of three vertices [23]. This characterization finds direct applications in corresponding fixed-parameter algorithms [7] (fixed-parameter tractability is also implied by a more general result of Leizhen Cai [3]), whereas our new characterization of 3-leaf powers requires a more tricky approach. Due to the lack of space, many proofs are deferred to the full version of this paper.

2 Preliminaries, Basic Definitions, and Previous Work

We consider only undirected graphs $G = (V, E)$ with $n := |V|$ and $m := |E|$. Edges are denoted as tuples (u, v) . For a graph $G = (V, E)$ and $u, v \in V$, let $d_G(u, v)$ denote the length of the shortest path between u and v in G . With $E(G)$, we denote the edge set E of a graph $G = (V, E)$. We call a graph $G' = (V', E')$ an *induced subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' = \{(u, v) \mid u, v \in V' \text{ and } (u, v) \in E\}$. An edge between two vertices of a cycle that is not part of the cycle is called *chord*. An induced cycle of length at least four is called *hole*. A *chordless graph* is a graph that contains no hole. For two sets A and B , $A \Delta B$ denotes the *symmetric difference* $A \setminus B \cup B \setminus A$.

Closely related to the well-known graph power concept (cf. [2–Section 10.6]) is the notion of a *k -leaf power* of a tree, introduced by Nishimura, Ragde, and Thilikos [21]:

Definition 1. Let T be a tree with vertex set V . The *k -leaf power* of T is the graph $T^k := (V, E)$ with $E := \{(u, v) \mid u, v \in V \text{ and } d_T(u, v) \leq k\}$.

The following problem is inspired by the problem of forming a phylogenetic tree¹ based on a binary similarity measure.

k -LEAF ROOT (LR k)

Instance: A graph G .

Question: Is there a tree T such that $T^k = G$?

Nishimura et al. [21] show that k -LEAF ROOT can be solved in polynomial time for $k \leq 4$. As already Nishimura et al. point out, in practice phylogenetic problems involve errors in distance estimators. This motivates the following.

CLOSEST k -LEAF ROOT (CLR k)

Instance: A graph $G = (V, E)$ and a nonnegative integer l .

Question: Is there a tree T such that T^k and G differ by at most l edges, that is, $|E(T^k) \Delta E(G)| \leq l$?

This problem is also denoted more precisely as CLR k EDGE EDITING. In this paper we also study two variations, where the distance estimator is assumed to have only one-sided errors:

- CLR k EDGE INSERTION: Only inserting edges into G is allowed to obtain T^k ;
- CLR k EDGE DELETION: Only deleting edges from G is allowed to obtain T^k .

CLR2 EDGE EDITING has been studied under various names in the literature. The first proof of its NP-completeness is due to Krivánek and Morávek [13], where it is called HIERARCHICAL-TREE CLUSTERING. Independently, the problem was studied by Shamir, Sharan, and Tsur as CLUSTER EDITING [23] and by Bansal, Blum, and Chawla as CORRELATION CLUSTERING [1]. CLR2 EDGE DELETION (also known as CLUSTER DELETION) was shown to be NP-complete by Natanzon [19].

¹ That is, a tree where leaves correspond to species and internal nodes represent evolutionary events.

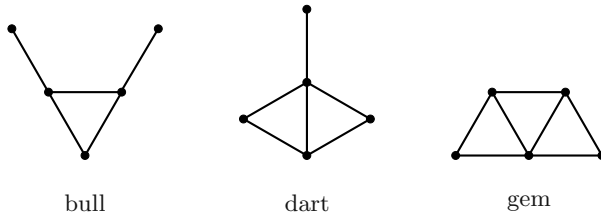


Fig. 1. 5-vertex graphs that occur as forbidden induced subgraphs

Lin, Kearney, and Jiang [16] consider a variant of k -LEAF ROOT where the inner nodes of the output tree are not allowed to have degree 2. They call this problem k -PHYLOGENETIC ROOT (PR k), and show that PR k can be solved in linear time for $k \leq 4$.² As for leaf roots, the generalization that allows for the input graph to contain errors is a better match for the biological motivation, and one can ask for the CLOSEST k -PHYLOGENETIC ROOT (CPR k), defined analogous to CLOSEST k -LEAF ROOT. CPR k is examined by Chen, Jiang, and Lin [4], who show that it is NP-complete for $k \geq 2$.

Among other things, we show that CLR3 is fixed-parameter tractable with respect to parameter l . That is, we show that CLR3 can be solved in $f(l) \cdot n^{O(1)}$ time, where f is an (exponential) function only depending on l . For small l , as might be naturally expected since l refers to the number of errors, efficient (polynomial-time) algorithms are possible. Two recent surveys on fixed-parameter tractability can be found in [6, 20].

3 Forbidden Subgraph Characterization for 3-Leaf Powers

It is not hard to see that graphs that are 2-leaf powers are exactly the graphs where every connected component is a clique. Shamir et al. [23] note that these graphs are characterized by a forbidden induced subgraph, namely a path of three vertices (P_3). In this section we derive a similar, but far less evident forbidden subgraph characterization of 3-leaf powers: they are chordal graphs that contain no induced bull, dart, or gem (see Figure 1).

Forbidden subgraph characterizations can be valuable in various ways. For instance, they can lead to fixed-parameter algorithms for the corresponding graph modification problems. Leizhen Cai [3] shows that with a finite set of forbidden subgraphs, finding the l edges to be modified is fixed-parameter tractable with respect to l . Using the single forbidden subgraph P_3 , this immediately applies to the case of 2-leaf powers; for 3-leaf powers, exploiting the subsequent forbidden subgraph characterization is one of the decisive ingredients of the fixed-parameter algorithms presented in Section 5. Note, however, that here Cai’s

² For $k = 4$, they show this only for connected graphs.

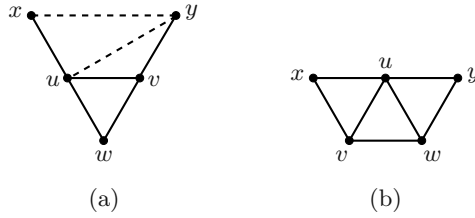


Fig. 2. Neighborhood of 3 vertices u, v, w from 3 different critical cliques

result does not apply directly, since chordal graphs do not admit a characterization by a finite set of forbidden subgraphs.

As we will see, 3-leaf powers are closely connected to the concept of a *critical clique*, which was introduced by Lin et al. [16].

Definition 2. A *critical clique* K of a graph G is a maximal module of G that is a clique. In other words, K is a maximal module of G such that K is a clique and $G \setminus K$ is a module of G .

In other words, a critical clique is a module that is maximal and a clique. The following connection to 3-leaf powers can be shown:

Lemma 1. Let G be a graph and K a critical clique of G . Then G is a 3-leaf power if and only if $G \setminus K$ is a 3-leaf power.

Lemma 2. Let G be a graph and K a critical clique of G . Then G is a 3-leaf power if and only if $G \setminus K$ is a 3-leaf power and K is a 3-leaf power.

(1) \Rightarrow (2): Let u, v, w be three vertices from K that belong to different critical cliques. We distinguish two cases.

- (a) There is a vertex x which is connected to exactly one of u, v, w , say to u (see Figure 2 (a)). Since v and w belong to different critical cliques, there must be a vertex y which is connected to only one of them, say to v . The edges (y, u) and (y, x) can be present or not, except that if (y, x) is present, then (y, u) must also be present, because otherwise x, y, v, u induce a hole. This leaves 3 possibilities, where we get the induced subgraphs bull, dart, and gem, respectively.
- (b) There is no vertex which is connected to exactly one of u, v, w (see Figure 2 (b)). Then there is a vertex x which is connected to exactly two of u, v, w , say to u and v (otherwise, u, v, w would have identical neighborhood, and would be in the same critical clique). Since u and v belong to different critical cliques, there is a vertex y which is adjacent to only one of them, say to u . By the precondition of this case, y is connected to w . The vertices x and y cannot be connected, since otherwise x, y, w, v induce a hole. We get an induced gem.

(2) ⇒ (1): Assume G contains a forbidden subgraph. Let u, v, w be the vertices of a triangle in the forbidden subgraph (in the case of the gem, the triangle which contains both degree-3 vertices). Then u, v, w form a clique. Let x and y be the remaining two vertices in the subgraph. Since each of u, v, w is adjacent to a different combination of x and y , they belong to 3 different critical cliques. □

Since between the vertices of two critical cliques either all pairwise or no connections are present, the concept of a *critical clique graph* [16] comes up naturally. As we will see, the structure of the critical clique graph is already close to the structure of the 3-leaf roots we are looking for. For easier distinction from the elements of G , we use the term *critical clique* for vertices in the critical clique graph.

Definition 3. Let $G = (V, E)$ be a graph. The *critical clique graph* $CC(G) = (C, E_C)$ is a graph where C is a partition of V into critical cliques K_i and E_C is the set of edges between critical cliques. Formally, $(K_i, K_j) \in E_C \iff \forall u \in K_i, v \in K_j : (u, v) \in E$.

Since every vertex of G belongs to exactly one critical clique, the critical clique graph of G can be constructed in $O(n \cdot m)$ time by iterating through the vertices and constructing the critical clique they are part of by comparing their neighborhood to that of all adjacent vertices.

The following lemma reveals details of the structure of a critical clique graph.

Lemma 3. Let G be a graph. Then $CC(G)$ is a forest.

Utilizing Lemmas 1, 2, and 3, we can obtain the main theorem of this section.

Theorem 1. Let G be a graph. Then the following are equivalent:
 (1) G is a leaf power.
 (2) G is chordal.

(1) ⇒ (2): If G is a leaf power, then G must be chordal [16]. Then, by Lemma 1 and Lemma 2, it does not contain any of the forbidden subgraphs.
 (2) ⇒ (1): If G is chordal, then so is $CC(G)$, since if $CC(G)$ contained a hole, we could also find a hole in G by taking one arbitrary vertex from each critical clique on the cycle. With Lemma 3, it follows that $CC(G)$ is a forest. For each connected component of $CC(G)$, construct a leaf root by attaching to each node a new leaf node for each vertex of the corresponding critical clique. Finally, create a new node and connect this node to an arbitrary inner node of each newly constructed tree. Then, the resulting tree T is a 3-leaf root of G . To see this, consider two vertices $u, v, u \neq v$ of G . They are connected in G iff they are in the same critical clique, or they are in two adjacent critical cliques. This is equivalent to the distance of u and v in T being 2 and 3, respectively. □

Table 1. Complexity of CLOSEST k -LEAF ROOT. The polynomial-time solvability of CLR2 EDGE INSERTION is trivial; the results for $k \geq 3$ are discussed in Section 4. The main new result is NP-completeness of CLR k EDGE INSERTION for $k \geq 3$

	$k = 2$	$k \geq 3$
Edge editing	NP-complete [13]	NP-complete
Edge deletion	NP-complete [19]	NP-complete
Edge insertion	P	NP-complete

4 NP-Completeness Results

In Table 1 we summarize known and new results on the classical complexity (P vs. NP) of CLOSEST k -LEAF ROOT problems. The NP-completeness of CLR k EDGE EDITING and CLR k EDGE DELETION for $k \geq 3$ can be shown by an adaption of the NP-completeness proof for CPR k by Chen et al. [4]. We refer to the full version of this paper for details on how to adapt their proof.

CLR2 EDGE INSERTION can be trivially solved in polynomial time, since it is exactly the problem of adding edges to a graph so that each connected component becomes a clique. However, it can be shown that for $k \geq 3$ this problem becomes NP-complete by giving a reduction from MAXIMUM EDGE BICLIQUE [22]; we defer the proof to the full version of this paper.

Theorem 2. CLR k EDGE INSERTION is NP-complete for $k \geq 3$

5 Fixed-Parameter Tractability Results for CLR3

In this section we show fixed-parameter tractability with respect to the number of editing operations l for CLR3 EDGE INSERTION, CLR3 EDGE DELETION, and CLR3 EDGE EDITING. According to the characterization of 3-leaf powers from Theorem 1, the algorithms have two tasks to fulfill:

- (1) Edit the input graph G to get rid of the forbidden subgraphs bull, dart, and gem.
- (2) Edit G to make it chordal.

Lin et al. [16] show the usefulness of the critical clique graph for the construction of the 3-leaf root (see also Section 3). The following lemma demonstrates that the critical clique graph is also of crucial importance for our algorithms solving CLR3: our algorithms work with the critical clique graph $CC(G)$ instead of G . We defer the proof of the lemma to the full version of this paper since it is similar to the proof of Lemma 2. We use C_4 to denote a chordless cycle of four vertices.

Lemma 4. Let G be a graph and C_4 a chordless cycle of four vertices. Then...

Following from Lemma 4, if we can show that there is an optimal solution for CLR3 problems which can be represented as editing operations on the critical clique graph, then the two above tasks can be fulfilled in two independent phases: First, eliminate each induced bull, dart, gem, and C_4 in the critical clique graph using a search tree of bounded depth. Then, edit each of the resulting critical clique graphs to make it a forest. The optimal solution is then the solution with minimum total number of editing operations in the two steps. The following two lemmas show that it is indeed correct to work only with $CC(G)$ instead of G .

Lemma 5. G C_4
 $CC(G)$ C_4

Lemma 6. G
 CLR3 EDGE EDITING $CC(G)$

Based on Lemmas 5 and 6, we can now consider a critical clique of G as a single vertex and work on $CC(G)$ instead of G . One more benefit of this approach is that we can eliminate several forbidden subgraphs in G by only one modification operation in $CC(G)$. A modification operation on $CC(G)$ can decrease the parameter l by more than one since it can correspond to more than one modification operation on G . Then, our algorithm scheme is as follows:

- (0) Construct $CC(G)$ from G .
- (1) Edit $CC(G)$ to get rid of the forbidden subgraphs bull, dart, gem, and C_4 .
- (2) Edit $CC(G)$ to make it a forest.

Note that after modifying $CC(G)$, two or more nodes in $CC(G)$ might obtain identical neighborhoods. Since each node in $CC(G)$ has to represent a critical clique in G , a merge operation is needed, which replaces these nodes in $CC(G)$ by a new node with the same neighborhood as the original nodes. Therefore, in the following, we assume that after each modification operation, we check for every pair of nodes whether a merge operation between them is possible, which can be done in $O(n \cdot m)$ time.

We now examine the running time of the respective steps. As mentioned in Section 3, $CC(G)$ can be constructed in $O(n \cdot m)$ time. By Cai’s result [3], there is a fixed-parameter algorithm for Step (1). More specifically, because of Lemma 4, we know that each triangle in $CC(G)$ is either part of a bull, dart, or gem, or it has one edge in common with a C_4 . We can then determine a forbidden subgraph by first finding a triangle in $O(n \cdot m)$ time, and then partitioning the $n - 3$ nodes not in the triangle into eight sets depending on to which of the three nodes in the triangle they are connected. This allows to find a bull, dart, gem, or C_4 in additional $O(n)$ time. If we do not find a triangle, we can find

a C_4 by determining the shortest cycle in $O(n \cdot m)$ time [9]. In summary, we find a forbidden subgraph in $O(n \cdot m)$ time.³

Therefore, we can employ a search tree which finds a forbidden subgraph and branches into several cases corresponding to each editing operation that destroys it. As an example, for edge deletion, this will lead to an $O(7^l \cdot nm)$ time algorithm, since the highest number of edges occurring in any forbidden subgraph is seven (gem). We mention in passing that the techniques applied by Gramm et al. [7, 8] for CLR2 could be adapted to improve the base 7 of the exponential component of the running time. In the descriptions of the respective algorithms, we now only need to deal with Step (2) to show fixed-parameter tractability.

As shown in the proof of Theorem 1, if $CC(G)$ has more than one connected component, we can solve the problem for each component independently, and then connect the generated leaf roots by adding a new inner node and connecting it to an arbitrary inner node of each leaf root. This allows us in the following without loss of generality to only consider connected graphs. Note that this property does not hold for CLOSEST k -PHYLOGENETIC ROOT, which makes it considerably harder to obtain analogous results.

5.1 Edge Deletion

As stated above, the task of Step (2) is to transform a bull-, dart-, gem-, and C_4 -free $CC(G)$, which does not contain a triangle as an induced subgraph, into a forest by edge deletions. Observe that after getting rid of all forbidden subgraphs in Step (1), throughout Step (2) $CC(G)$ always remains bull-, dart-, gem-, and C_4 -free when only edge deletions are allowed. Hence, Step (2) of our algorithm scheme can be handled with a polynomial-time algorithm, as stated in the following lemma.

Lemma 7. *Let G be a graph with n vertices and m edges. Let $CC(G)$ be the connected component graph of G . If $CC(G)$ is bull-, dart-, gem-, and C_4 -free, then there is a polynomial-time algorithm, CLR3 EDGE DELETION, that finds a maximum weight spanning tree of G in $O(m \log n)$ time.*

Theorem 3. *CLR3 EDGE DELETION runs in $O(7^l \cdot nm)$ time, where l is the height of the search tree.*

We employ a search tree of height bounded by l . In each inner node of the search tree, we find a forbidden subgraph and branch into at most seven cases corresponding to the edges of the forbidden subgraph. At each leaf of the search tree, we find a maximum weight spanning tree in $O(m \log n)$ time. In summary, we have a running time of $O(7^l \cdot nm)$. □

³ Note that using algorithms based on matrix multiplication, we can alternatively do this in $O(n^{2.38})$ time [5].

5.2 Edge Insertion

If $CC(G)$ after Step (1) contains no cycle, i.e., it is a tree, then there is no edge insertion required. For a $CC(G)$ containing at least one cycle, the only possible way to make it a tree by edge insertions is to trigger a merge operation for some nodes on this cycle such that the remaining nodes induce no cycle. Recall that two nodes can be merged iff they are adjacent and they have the same neighborhood. Thus, in order to merge two nodes K_i and K_j , we have to insert an edge between them if they are not already adjacent; furthermore, we need to connect K_i to all neighbors of K_j and connect K_j to all neighbors of K_i . Since each cycle of $CC(G)$ has length at least four, two nodes K_i and K_j on the same cycle either are not adjacent or there are at least two neighbors which are not common to K_i and K_j . Hence, we need at least one edge insertion to merge two nodes on a cycle.

We show the fixed-parameter tractability of CLR3 EDGE INSERTION with respect to l by giving a simple search tree algorithm that tries all possible pairs of nodes to merge in a cycle. For this, it suffices to determine an upper bound for the length of a cycle in $CC(G)$ that depends only on l . We achieve this by giving a connection between the triangulation of a hole and the merge operations that turn a cycle into a tree.

A *triangulation* of a hole $C = (V_C, E_C)$, where V_C denotes the set of the vertices on this cycle and E_C the set of the edges, is a set D of chords of C such that there is no hole in $C' = (V_C, E_C \cup D)$. A triangulation F of a graph G is *minimal* if no proper subset of F triangulates G .

Lemma 8. *Let C be a cycle of length n and D a triangulation of C . Then $|D| = n - 3$.*

Kaplan, Shamir, and Tarjan [11] show that a minimal triangulation D of an n -cycle C consists of $n - 3$ chords, which implies that a graph G that can be triangulated by at most l edge insertions cannot have a chordless cycle of length more than $l + 3$. This is also the key idea of one of their fixed-parameter algorithms for MINIMUM FILL-IN, which is the problem to make a graph chordal by edge insertion. With Lemma 8, we conclude that the maximum cycle length of $CC(G)$ is bounded above by $l + 3$; otherwise, there is no solution to CLR3 EDGE INSERTION using only l insertion operations.

Altogether, we get the following theorem.

Theorem 4. CLR3 EDGE INSERTION is fixed-parameter tractable with respect to l . Given a graph $G = (V, E)$ and an integer l , we can decide in time $O(n^{l+3})$ whether G can be made chordal by at most l edge insertions.

5.3 Edge Editing

In this section we extend the algorithm for CLR3 EDGE INSERTION from Section 5.2 to solve CLR3 EDGE EDITING by additionally taking edge deletions into account. We distinguish two types of cycles: the *long* cycles having length greater than $l + 3$, and the *short* cycles having length at most $l + 3$.

We can destroy a short cycle in $CC(G)$ by deleting at least one edge from it, or by merging some critical cliques. This means we have at most $l + 3$ possible edge deletions and at most $(l + 3)^2$ possible merge operations. However, merge operations with both edge deletion and edge insertion are more complicated than merge operations with only edge insertion. Suppose that we merge a pair of critical cliques K_i and K_j on a cycle. As with only edge insertions allowed, we insert an edge between K_i and K_j if they are not adjacent. There may be some critical cliques which are neighbors of K_i but not of K_j or vice versa. To satisfy the neighborhood condition of a critical clique, for each of these neighbors which are not common to K_i and K_j , we have to either insert an edge to make it a common neighbor of both critical cliques, or delete an edge to make it nonadjacent to both critical cliques. However, there may be at most l such non-common neighbors, since there are at most l edge editing operations allowed. A merge operation between K_i and K_j is then possible only if they have at most l noncommon neighbors. Thus, we have at most 2^l different ways to merge these two critical cliques. Altogether, we now have $(l + 3) + (l + 3)^2 \cdot 2^l$ branchings to transform a short cycle into a tree.

Theorem 5. CLR3 EDGE EDITING $\dots G = (V, E) \dots l \dots$

6 Concluding Remarks

Our algorithmic results fall into the broad category of complexity for graph modification problems. In addition, we recently obtained a fixed-parameter tractability result for CLR3 VERTEX DELETION, the NP-complete problem that asks for the least number of vertices to delete to make a graph a 3-leaf root. The line of research initiated in our work offers several future challenges. We only mention four points.

- In ongoing work we examine the generalization of our fixed-parameter tractability results for CLOSEST 3-LEAF ROOT to CLOSEST 4-LEAF ROOT. For $k \geq 5$ the question is completely open. The difficulty here lies in the more complicated structure of the critical clique graph; for example, it is no longer required to be a tree.
- It remains open to provide a problem kernel for 3-LEAF ROOT [6, 20].
- One challenge is to investigate whether similar fixed-parameter tractability results can be achieved for the closely related phylogenetic root problems studied in [4, 16]. Forbidding degree-2 nodes there in the output trees seems to make things more elusive, though.
- From a more applied point of view, it would be interesting to see how small the combinatorial explosion for CLR3 and its variants in the parameter l (denoting the number of modifications) can be made. Encouraging results for the “simpler” but still NP-complete CLOSEST 2-LEAF ROOT problem are obtained in [7, 8] (where the problem is referred to as CLUSTER EDITING).

References

1. N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *Proc. 43rd FOCS*, pages 238–247. IEEE Computer Society, 2002.
2. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.
3. L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58:171–176, 1996.
4. Z.-Z. Chen, T. Jiang, and G. Lin. Computing phylogenetic roots with bounded degrees and errors. *SIAM Journal on Computing*, 32(4):864–879, 2003.
5. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.
6. M. R. Fellows. New directions and new challenges in algorithm design and complexity, parameterized. In *Proc. 8th WADS*, volume 2748 of *LNCS*, pages 505–520. Springer, 2003.
7. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. In *Proc. 5th CIAC*, volume 2653 of *LNCS*, pages 108–119. Springer, 2003. Long version to appear in *Theory of Computing Systems*.
8. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.
9. A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
10. T. Jiang, G. Lin, and J. Xu. On the closest tree k th root problem. Manuscript, Department of Computer Science, University of Waterloo, 2000.
11. H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.
12. P. E. Kearney and D. G. Corneil. Tree powers. *Journal of Algorithms*, 29(1):111–131, 1998.
13. M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.
14. L. C. Lau. Bipartite roots of graphs. In *Proc. 15th ACM-SIAM SODA*, pages 952–961. ACM/SIAM, 2004.
15. L. C. Lau and D. G. Corneil. Recognizing powers of proper interval, split, and chordal graphs. *SIAM Journal on Discrete Mathematics*, 18(1):83–102, 2004.
16. G. Lin, P. E. Kearney, and T. Jiang. Phylogenetic k -root and Steiner k -root. In *Proc. 11th ISAAC*, volume 1969 of *LNCS*, pages 539–551. Springer, 2000.
17. Y. L. Lin and S. S. Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8(1):99–118, 1995.
18. R. Motwani and M. Sudan. Computing roots of graphs is hard. *Discrete Applied Mathematics*, 54(1):81–88, 1994.
19. A. Natanzon. Complexity and approximation of some graph modification problems. Master’s thesis, Department of Computer Science, Tel Aviv University, 1999.
20. R. Niedermeier. Ubiquitous parameterization—invitation to fixed-parameter algorithms. In *Proc. 29th MFCS*, volume 3153 of *LNCS*, pages 84–103. Springer, 2004.
21. N. Nishimura, P. Ragde, and D. M. Thilikos. On graph powers for leaf-labeled trees. *Journal of Algorithms*, 42(1):69–108, 2002.

22. R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.
23. R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. In *Proc. 28th WG*, volume 2573 of *LNCS*, pages 379–390. Springer, 2002. Long version to appear in *Discrete Applied Mathematics*.

On Compact and Efficient Routing in Certain Graph Classes (Extended Abstract)

Feodor F. Dragan¹ and Irina Lomonosov²

¹ Department of Computer Science, Kent State University, Kent, Ohio, USA
dragan@cs.kent.edu

² Department of Computer Science, Hiram College, Hiram, Ohio, USA
lomonosovi@hiram.edu

Abstract. In this paper we refine the notion of tree-decomposition by introducing acyclic (R, D) -clustering, where clusters are subsets of vertices of a graph and R and D are the maximum radius and the maximum diameter of these subsets. We design a routing scheme for graphs admitting induced acyclic (R, D) -clustering where the induced radius and the induced diameter of each cluster are at most 2. We show that, by constructing a family of special spanning trees, one can achieve a routing scheme of deviation $\Delta \leq 2R$ with labels of size $O(\log^3 n / \log \log n)$ bits per vertex and $O(1)$ routing protocol for these graphs. We investigate also some special graph classes admitting induced acyclic (R, D) -clustering with induced radius and diameter less than or equal to 2, namely, chordal bipartite, homogeneously orderable, and interval graphs. We achieve the deviation $\Delta = 1$ for interval graphs and $\Delta = 2$ for chordal bipartite and homogeneously orderable graphs.

1 Introduction

Routing is one of the basic tasks that a distributed network of processors must be able to perform. A routing scheme is a mechanism that can deliver packets of information from any node of the network to any other node. More specifically, a routing scheme is a distributed algorithm. Each processor in the network has a routing daemon (known also as a routing algorithm or a routing procedure) running on it. This daemon receives packets of information and has to decide whether these packets have already reached their destination, and if not, how to forward them towards their destination. A network can be viewed as a graph, with the vertices representing processors and the edges representing direct connections between processors. It is naturally desirable to route messages along paths that are as short as possible.

Routing scheme design is a well-studied subject. For a general overview we refer the reader to [14]. Most routing schemes are local that assign two kind of labels to every vertex of a graph. The first label is the address of the vertex, the second is a data structure called routing table. These labels are assigned in such a way that at every source vertex x its routing daemon can quickly decide, based on the two labels stored locally in x and the address of

any destination node y , whether the packet has reached its destination, and if not, to which neighbor of x to forward the packet.

A straightforward approach to routing is to store a routing table at each vertex of the graph, specifying for each destination y the first edge (or identifier of that edge, indicating the output port) along some shortest path from x to y . While this approach guarantees optimal (shortest path) routing, it is too expensive for large systems since it requires total $O(n^2 \log \delta)$ memory bits for an n -vertex graph with maximum degree δ . Thus, for large scale communication networks, it is important to design routing schemes that produce short enough routes and have sufficiently low memory requirements.

Unfortunately, for every shortest path routing strategy and for all δ , there is a graph of degree bounded by δ for which $\Omega(n \log \delta)$ bit routing tables are required simultaneously on $\Theta(n)$ vertices [11]. This matches the memory requirements of complete routing tables. To obtain routing schemes for general graphs that use $o(n)$ of memory at each vertex, one has to abandon the requirement that packets are always delivered via shortest paths, and settle instead for the requirement that packets are routed on paths that are relatively close to shortest. The efficiency of a routing scheme is measured in terms of its additive stretch, called *additive stretch* (or multiplicative stretch, called *multiplicative stretch*), namely, the maximum surplus (or ratio) between the length of a route, produced by the scheme for a pair of vertices, and the shortest route. There is a tradeoff between the memory requirements of a routing scheme and the worst case stretch factor it guarantees. Any multiplicative t -stretched routing scheme must use $\Omega(\sqrt{n})$ bits for some vertices in some graphs for $t < 5$ [18], $\Omega(n)$ bits for $t < 3$ [9], and $\Omega(n \log n)$ bits for $t < 1.4$ [11]. These lower bounds show that it is not possible to lower memory requirements of a routing scheme for an arbitrary network if it is desirable to route messages along paths close to optimal. Therefore it is interesting, both from a theoretical and a practical view point, to look for specific routing strategies on graph families with certain topological properties.

One way of implementing such routing schemes, called *interval routing*, has been introduced in [16] and later generalized in [13]. In this special routing method, the complete routing tables are compressed by grouping the destination addresses which correspond to the same output port. Then each group is encoded as an interval, so that it is easy to check whether a destination address belongs to the group. This approach requires $O(\delta \log n)$ bit labels and $O(\log \delta)$ forwarding protocol, where δ is the maximum degree of a vertex of the graph. A graph must satisfy some topological properties in order to support interval routing, especially if one insists on paths close to optimal. Routing schemes for many graph classes were obtained by using interval routing techniques. The classical and most recent results in this field are presented in [8].

New routing schemes for interval graphs, circular-arc graphs and permutation graphs were presented in [5]. The design of these simple schemes uses properties of intersection models. Although this approach gives some improvement over existing earlier routing schemes, the local memory requirements increase with the degree of the vertex as in interval routing.

Graphs with regular topologies, as hypercubes, tori, rings, complete graphs, etc., have specific routing schemes using $O(\log n)$ -bit labels. It is interesting to investigate which other classes of graphs admit routing schemes with labels not depending on vertex degrees, that route messages along near-optimal path. A shortest path routing scheme for trees of arbitrary degree and diameter is described in [7, 17]. It assigns each vertex of an n -vertex tree a $O(\log^2 n / \log \log n)$ -bit label. Given the label of a source vertex and the label of a destination vertex it is possible to determine in constant time the neighbor of the source vertex that leads towards the destination. These routing schemes for trees serve as a base for designing routing strategies for more general graphs. Indeed, if there is a family of spanning trees such that for each pair of vertices of a graph, there is a tree in the family containing a low-stretch path between them, then the tree routing scheme can be applied within that tree. This approach was used in [4] to obtain a routing scheme of deviation 2 with labels of size $O(\log^3 n / \log \log n)$ bits per vertex and $O(1)$ routing protocol for chordal graphs. The scheme uses the notion of tree-decomposition introduced in [15]. There, a family of spanning trees is a collection of Breadth-First-Search trees associated with each node of the tree-decomposition. It is shown that, despite the fact that the size of the family can be $O(n)$, it is enough for each vertex to keep routing labels of only $O(\log n)$ trees and, nevertheless, for each pair of vertices, a tree containing a low-stretch path between them can be determined in constant time.

In this paper we refine the notion of tree-decomposition by introducing acyclic (R, D) -clustering, where clusters are subsets of vertices of a graph and R and D are the maximum radius and diameter of these subsets. We develop a routing scheme for graphs admitting induced acyclic (R, D) -clustering where the induced radius and the induced diameter of each cluster are at most 2. We show that, by constructing a family of special spanning trees, one can produce a routing scheme of deviation $\Delta \leq 2R$ with labels of size $O(\log^3 n / \log \log n)$ bits per vertex and $O(1)$ routing protocol for these graphs. Our routing strategy is inspired by and based on the work of Dourisboure and Gavoille [4]. Recently we learned that [3], too, generalizes the approach taken in [4] and obtains a routing scheme of deviation $\Delta \leq 2D$ with labels of size $O(D \log^3 n)$ bits per vertex and $O(\log(D \log n))$ routing protocol for the so-called tree-length D graphs [3] (which turns out to be equivalent to the class of graphs admitting acyclic (D, D) -clustering).

We investigate some special graph classes admitting induced acyclic (R, D) -clustering with induced radius and diameter less than or equal to 2, namely, chordal bipartite, homogeneously orderable, and interval graphs. We achieve the deviation $\Delta = 1$ for interval graphs and $\Delta = 2$ for chordal bipartite and homogeneously orderable graphs, while the routing schemes of [3, 4] produce $\Delta = 2$ for interval graphs and $\Delta = 4$ for chordal bipartite graphs. To the best of our knowledge this is the first routing scheme that is presented for homogeneously orderable graphs. Note that they include such well known families of graphs as distance-hereditary graphs, strongly chordal graphs, dually chordal graphs as well as homogeneous graphs (see [2]). Additionally, we achieve a constant time routing protocol and slightly lower memory requirements for chordal bipartite

graphs (from [3] one could infer for chordal bipartite graphs a scheme with labels of size $O(\log^3 n)$ bits per vertex and $O(\log \log n)$ routing protocol).

2 Preliminaries

All graphs occurring in this paper are connected, finite, undirected, loopless, and without multiple edges. For a subset $S \subseteq V$ of vertices of G , let $G(S)$ be a subgraph of G induced by S . By $n = |V|$ we denote the number of vertices in G . The distance $dist_G(u, v)$ between vertices u and v of a graph $G = (V, E)$ is the smallest number of edges in a path connecting u and v . The distance between a vertex $u \in V$ and a set S is $dist_G(u, S) = \min_{v \in S} \{dist_G(u, v)\}$. The radius of a set S in G is $rad_G(S) = \min_{v \in S} \{max_{u \in S} \{dist_G(v, u)\}\}$ and the diameter is $diam_G(S) = \max_{v, u \in S} \{dist_G(v, u)\}$. The radius of a set S is $rad(S) = \min_{v \in S} \{max_{u \in S} \{dist_{G(S)}(v, u)\}\}$ and the diameter is $diam(S) = \max_{v, u \in S} \{dist_{G(S)}(v, u)\}$. A vertex $v \in S$ such that $dist_{G(S)}(u, v) \leq rad(S)$ for any $u \in S$, is called a center of S . Also, we denote by $N_G(v) = \{u \in V : uv \in E\}$ the neighborhood of a vertex v in G and by $N_G[v] = N_G(v) \cup \{v\}$ the closed neighborhood of v in G . The k -neighborhood $N^k(v)$ of a vertex v of G is the set of all vertices of distance k to v : $N^k_G(v) = \{u \in V : dist_G(u, v) = k\}$.

Our concept of acyclic (R, D) -clustering is a tree decomposition introduced by Robertson and Seymour [15], except that clusters have to satisfy bounds on the radius and the diameter.

Definition 1. Let $G = (V, E)$ be a graph and $T = (C_1, \dots, C_\kappa)$ an (R, D) -clustering of G . Then $\mathcal{C} = \{C_1, C_2, \dots, C_\kappa\}$ is a (R, D) -clustering of V if

$$\begin{aligned} & \bigcup_{C \in \mathcal{C}} C = V, \\ & \text{if } uv \in E, \text{ then } \exists C \in \mathcal{C} \text{ such that } u, v \in C, \\ & \text{if } X, Y, Z \in \mathcal{C}, \text{ then } Y \cap X \neq \emptyset \implies X \cap Z \subseteq Y, \\ & \max_{C \in \mathcal{C}} \{rad_G(C)\} \leq R \text{ and } \max_{C \in \mathcal{C}} \{diam_G(C)\} \leq D, \text{ and } R \geq D \end{aligned}$$

T is called a (R, D) -clustering of G . The value $\kappa = |\mathcal{C}|$ is called the number of clusters, R and D are called the radius and the diameter, respectively. We assume that acyclic clustering is minimal, meaning that no cluster is contained in any other cluster (clearly any acyclic clustering can be reduced).

We say that a graph $G = (V, E)$ admits an induced acyclic (R, D) -clustering if $\max_{C \in \mathcal{C}} \{rad(C)\} \leq R$ and $\max_{C \in \mathcal{C}} \{diam(C)\} \leq D$, where R and D are non-negative integers called the radius and the diameter, respectively. An example of a graph admitting an induced acyclic $(1,2)$ -clustering is given in Fig. 1.

Note that the notion of acyclic (R, D) -clustering and the well-known notion of tree-width of a graph (see [15]), are not related to each other. For instance, any clique has an acyclic $(1,1)$ -clustering but is of unbounded tree-width, whereas

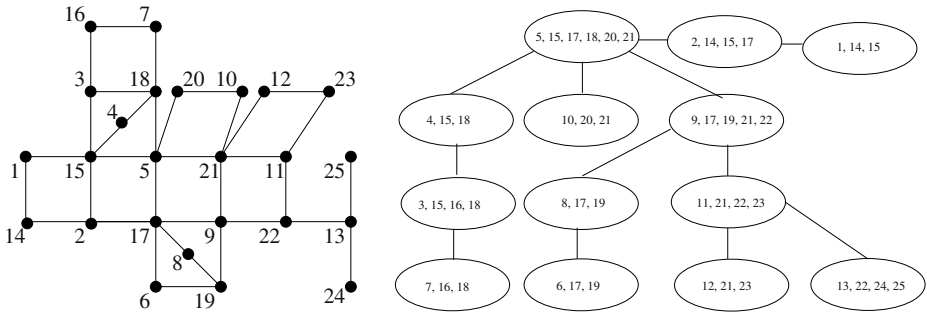


Fig. 1. A graph admitting an induced acyclic (1,2)-clustering and its tree-decomposition

any cycle of length k has tree-width 2 but admits only acyclic $(\Omega(k), \Omega(k))$ -clustering.

We will use the following property of acyclic clustering. It is proved in the full version of the paper [6]. Recall that a graph is chordal if it does not contain any induced cycles of length greater than 3. A vertex v is simplicial in G if $N_G(v)$ is a clique in G .

Lemma 1.

$$\begin{array}{l}
 G = (V, E) \text{ admits an acyclic } (R, D)\text{-clustering} \\
 \iff G^+ = (V, E^+) \text{ admits an acyclic } (R, D)\text{-clustering} \\
 \iff G^+ \text{ is chordal and } \text{diam}_G(X) \leq D \\
 \iff \text{rad}_G(X) \leq R
 \end{array}$$

Since a chordal graph can have at most n maximal cliques [12], from Lemma 1 we obtain that any acyclic (R, D) -clustering has at most n clusters, i.e., $\kappa \leq n$.

3 Routing Scheme

Let G be a graph that admits an acyclic (R, D) -clustering and T be a tree-decomposition associated with it. We assume that T is rooted (say, at C_1). In a rooted tree T , $nca_T(X, Y)$ denotes the nearest common ancestor of nodes X and Y of T .

Definition 2.

Let $u \in G$ be a node of G . Let $B(u)$ be the subtree of T rooted at u and containing all nodes v of T such that $u \in Z$.

It is well known that any tree T with κ nodes has a node C , called a centroid, which can be found in $O(\kappa)$ time, such that any maximal by inclusion subtree of T , not containing C , (i.e., any connected component of $T \setminus C$) has at most $\kappa/2$ nodes. For the tree T of acyclic clustering we build a hierarchical tree H recursively as follows. All nodes of T are nodes in H . The root of H is C , a centroid of T , and

the children are the roots of the hierarchical trees of the connected components of $T \setminus C$. Note that the height of H is $O(\log \kappa)$. The hierarchical tree for the graph in Fig. 1 is given in Fig. 2.

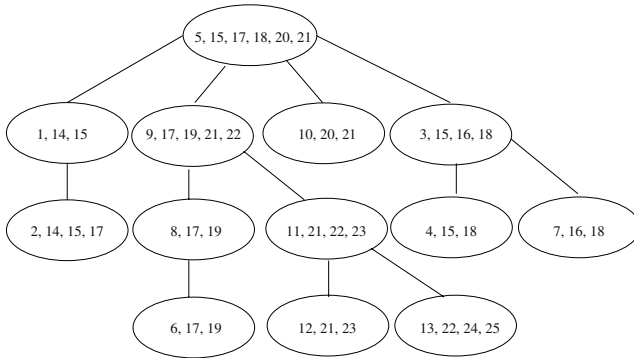


Fig. 2. A hierarchical tree H for the graph in Fig. 1

Let $G = (V, E)$ be a graph that admits an induced acyclic (R, D) -clustering with $R \leq 2$ and $D \leq 2$. Let X be a node of H and u be a vertex of G such that $u \notin X$ and $B(u)$ is a descendant of X in H . Let $P = \{u = z_0, z_1, z_2, \dots, z_k = x^*\}$, $x^* \in X$, be a shortest path of G from u to X .

Let $C_0 = B(u)$ and C_i be the cluster closest to C_{i-1} in T such that $z_{i-1}, z_i \in C_i$, $1 \leq i \leq k$. Note that such clusters exist by condition 2 of Definition 1. Let Q_i be the shortest path in T between C_{i-1} and C_i , $1 \leq i \leq k$. Let Q_{k+1} be the shortest path in T between C_k and X . Observe that, by condition 3 of Definition 1, $z_{i-1} \in Y$ for all $Y \in Q_i$. Let $Q(P) = \bigcup_{i=1}^{k+1} Q_i$ be a path between $B(u)$ and X and Q' be the shortest path between $B(u)$ and X in T . Note that, in general, $Q(P)$ is not a simple path, and $Q' \subseteq Q(P)$ for any path P between u and X .

For any two clusters Y and Z such that $Y, Z \in Q'$, we say that Y precedes Z , denoted by $Y \prec Z$, if Y is closer to $B(u)$ in T than Z . We use a notation $Y \preceq Z$ if $Y \prec Z$ or $Y = Z$.

Lemma 2. For any path $P = \{u = z_0, z_1, z_2, \dots, z_k = x^*\}$ between u and X in G , $Q(P) = Q'$.

Obviously, $C_0 = B(u) \in Q'$ for any P . Assume, by induction, that there exists a path $P = \{z_0, z_1, z_2, \dots, z_k\}$ between u and X such that $C_l \in Q'$ for $0 \leq l \leq i-1 < k$ and $C_0 \preceq C_1 \preceq \dots \preceq C_{i-1}$. We will show that there exists a path P' between u and X such that $C'_l \in Q'$ for $0 \leq l \leq i$ and $C'_0 \preceq C'_1 \preceq \dots \preceq C'_{i-1} \preceq C'_i$ as follows. Let C be a cluster closest to C_i in T such that $C \in Q'$. Since $X \in Q'$, there exists an integer p such that $i < p \leq k+1$ and $C \in Q_p$. Let $j > 0$ be the smallest number such that $C \in Q_{i+j}$. Notice that $z_{i+j-1} \in C$. Since $z_{i-1} \in C$ and C has diameter 2, we immediately obtain that $1 \leq j \leq 2$.

Note that $z_i \notin C$, otherwise $C = C_i \in Q'$, a contradiction with $C_i \notin Q'$. Thus, $j = 2$, and C contains z_{i+1} .

We claim that $C_{i-1} \prec C$. Otherwise, C_{i-1} would contain either z_i , meaning $C_i = C_{i-1} \in Q'$, a contradiction, or z_q , $q > i$, which is not possible, since C_{i-1} has diameter 2 and P is a shortest path. Let C^* be the cluster closest to C_{i-1} in T such that $z_{i+1} \in C^*$. Since $z_{i+1} \notin C_{i-1}$, we have $C_{i-1} \prec C^* \preceq C$. Since C^* is on the path in T between C_{i-1} and C_i , by condition 3 of Definition 1, $z_{i-1} \in C^*$. Recall that P is a shortest path and, therefore, z_{i-1} and z_{i+1} are not adjacent in G . Since C^* has induced diameter 2, there exists a vertex $z^* \in C^*$ such that z^* is adjacent to both z_{i-1} and z_{i+1} . We replace z_i with z^* in P and obtain a new shortest path $P' = \{z_0, z_1, \dots, z_{i-1}, z^*, z_{i+1}, \dots, z_k\}$. Clearly, the paths $Q(P')$ and $Q(P)$ have a common prefix $\bigcup_{i=1}^{i-1} Q_i$.

Let C'_i be the cluster closest to C_{i-1} such that $z_{i-1}, z^* \in C'_i$. We will prove that $C'_i \in Q'$ and $C_{i-1} \preceq C'_i$ as follows. Assume, by contradiction, that $C'_i \notin Q'$. Let C''_i be the cluster closest to C'_i in T such that $C''_i \in Q'$. Since C''_i is on the path in T between C'_i and C^* , $z^* \in C'_i$ and $z^* \in C^*$, by condition 3 of Definition 1, $z^* \in C''_i$. Similarly, since C''_i is on the path in T between C'_i and C_{i-1} , $z_{i-1} \in C'_i$ and $z_{i-1} \in C_{i-1}$, by condition 3 of Definition 1, $z_{i-1} \in C''_i$. Obviously, C''_i is closer to C_{i-1} than C'_i . Since $z^*, z_{i-1} \in C''_i$, we obtain a contradiction, which proves $C'_i \in Q'$.

It remains to prove that $C_{i-1} \preceq C'_i$. Consider other possibilities. If $C_{i-2} \prec C'_i \prec C_{i-1}$, then, by condition 3 of Definition 1, $z_{i-2} \in C'_i$. In this case, C'_i is the cluster containing z_{i-2} and z_{i-1} and closer to C_{i-2} than C_{i-1} , a contradiction. If $C'_i \preceq C_{i-2}$, then C'_i contains a vertex z_{i-j} , $j > 2$, which is not possible since $z^* \in C'_i$, C'_i has diameter 2, and P is a shortest path. Thus, $C_{i-1} \preceq C'_i$. \square

In the full version of this paper [6], we show that such a path P , which we call a Q -path, can be constructed in $O(n^2)$ time.

Lemma 3. *Let P be a Q -path in T and $W = \{w \in P : B(w) \notin Q' = Q(P)\}$. Then $|W| \leq 3$.*

By Lemma 2, for each $w \in P$ there exists a cluster $C_w \in Q'$ such that $w \in C_w$. By Definition 2, for $w \in W$, $B(w) \in Q^*$ holds, where Q^* is the path between C_w and the root of T . Clearly, $B(w) \notin Q' \cap Q^*$, otherwise $B(w) \in Q'$. Thus, $B(w)$ is on the path between $nca_T(B(u), X)$ and the root of T . Since $w \in C_w$, $w \in B(w)$, and $nca_T(B(u), X)$ is on the path between C_w and $B(w)$, by condition 3 of Definition 1, $w \in nca_T(B(u), X)$ for all $w \in W$. Since the diameter of clusters is 2, and P is a shortest path, $|P \cap nca_T(B(u), X)| \leq 3$. Thus, $|W| \leq 3$. \square

Corollary 1. *Let P be a Q -path in T and $z \in P$. Let $B(z) \in Q' = Q(P)$. Then $nca_T(B(z), X) \in G$.*

Let z be a vertex of P . Assume that $B(z) \in Q' = Q(P)$ and consider possible arrangements of nodes X , $B(u)$, and $B(z)$ in H , taking into account that X is an ancestor of $B(u)$ in H . First, note that $B(z)$ cannot be an ancestor

of X in H , otherwise during the construction of hierarchical subtree rooted at $B(z)$, $B(u)$ and X would belong to different connected components of $T \setminus \{B(z)\}$, and, therefore, X could not be an ancestor of $B(u)$ in H . Second, if there exists a node Y such that X and $B(z)$ are descendants of Y in H , then $Y \in Q'$, and, again, during the construction of hierarchical subtree rooted at Y , $B(u)$ and X would belong to different connected components of $T \setminus \{Y\}$, and, therefore, X could not be an ancestor of $B(u)$ in H . Thus, if $B(z) \in Q'$, then the only possible arrangement is that $B(z)$ is a descendant of X in H . If $B(z) \notin Q'$, then, by Lemma 3, the number of such vertices z is bounded by 3. \square

Lemma 4. *Let $u, v \in V(G)$ and $X = nca_H(B(u), B(v))$. Let Q'_{uv} be the shortest path between u and v in G .*

Let $P = \{u = z_0, z_1, z_2, \dots, z_k = v\}$ be a path from u to v and $C_0 = B(u)$. Let C_i be the cluster closest to C_{i-1} in T such that $z_{i-1}, z_i \in C_i, 1 \leq i \leq k$. Note that such clusters exist by condition 2 of Definition 1. Let Q_i be the shortest path in T between C_{i-1} and $C_i, 1 \leq i \leq k$. Let Q_{k+1} be the shortest path in T between C_k and $B(v)$. Let $Q_{uv}(P) = \bigcup_{i=1}^{k+1} Q_i$ be a path between $B(u)$ and $B(v)$ and Q'_{uv} be the shortest path between $B(u)$ and $B(v)$ in T . Note that $Q'_{uv} \subseteq Q_{uv}(P)$ for any path P between u and v . By condition 3 of Definition 1, $z_{i-1} \in Y$ for all $Y \in Q_i, 1 \leq i \leq k+1$. Thus, any node of $Q_{uv}(P)$ and, hence, any node of Q'_{uv} contains a vertex of any path P between u and v . By construction of $H, X \in Q'_{uv}$ and, therefore, X is a separator between u and v . \square

For any node X of H , we construct a tree in G in the following way. Let U be a set of vertices of G such that $U \subseteq \{V \setminus X\}$ and $B(u)$ is a descendant of X in H for $u \in U$. First, for each $u \in U$, we construct a Q -simple shortest path $P(u)$ from u to X . Second, we construct a tree $t(X)$ spanning X such that its diameter $diam_{t(X)} = \max_{x_1, x_2 \in X} \{dist_t(x_1, x_2)\}$ is minimal. Clearly, $diam_{t(X)} \leq 2R$. Finally, we build a graph $G_X = \bigcup_{u \in U} P(u) \cup t(X)$ and construct in a Breadth-First-Search manner starting from $t(X)$ a special spanning tree \mathcal{T} of G_X .

Lemma 5. *Let \mathcal{T} be a spanning tree of G_X constructed as above. Let $u \in U$ and X be a node of H . Let $P(u)$ be a path in G_X from u to X and $W(P(u)) = \{z \in P(u) : B(z) \text{ is not a descendant of } X \text{ in } H\}$.*

Let $L_i = \{v \in V(G_X) : dist_{G_X}(v, X) = i\}, i \geq 0$, be the BFS-layers of G_X with respect to X . A spanning tree \mathcal{T} of G_X can be constructed starting from $t(X)$ in the following way. For all $u \in L_1$, the $parent(u)$ is a vertex $x \in X$ such that $|W(P(u))|$ is minimum, where $P(u)$ is the path $\{u, x\}$ of G_X . For all $u \in L_i, i > 1$, $parent(u)$ is a neighbor $v \in L_{i-1}$ of u in G_X such that $|W(P(u))|$ is minimum, where $P(u) = \{u, P(v)\}$. The above construction guarantees that u is connected to X in \mathcal{T} via a path $P(u)$ with minimum possible $|W(P(u))|$. Since there is a path in G_X between $u \in U$ and X that is Q -simple, by Corollary 1, $|W(P(u))| \leq 3$ for any $u \in U$. \square

Lemma 6. Let $u, v \in G$, $X = nca_H(B(u), B(v))$, \mathcal{T} be a tree rooted at X , $P_{\mathcal{T}}$ be the path from u to v in \mathcal{T} , $z \in P_{\mathcal{T}}$ be a vertex on the path between u and v in \mathcal{T} such that $B(z) \in X$. Then

By Lemma 5, there are at most 3 such vertices on the path between u and X and there are at most 3 more such vertices on the path between v and X . Since X has induced diameter 2, there is at most 1 other such vertex of X that is on the path between u and v in \mathcal{T} . \square

Lemma 7. Let $u, v \in G$, $X = nca_H(B(u), B(v))$, \mathcal{T} be a tree rooted at X , P_G be the path from u to v in G , $P_{\mathcal{T}}$ be the path from u to v in \mathcal{T} . Then $dist_{\mathcal{T}}(u, v) \leq dist_G(u, v) + \Delta$, where $\Delta \leq diam_t(X)$.

By Lemma 4, X is a separator between u and v . Let P_G be a shortest path from u to v in G . Let $u' \in P_G$ be the vertex closest to u such that $u' \in X$ and let $v' \in P_G$ be the vertex closest to v such that $v' \in X$. Clearly, $dist_G(u, v) = dist_G(u, u') + dist_G(u', v') + dist_G(v', v)$. Similarly, let $P_{\mathcal{T}}$ be the path from u to v in \mathcal{T} . Let $u'' \in P_{\mathcal{T}}$ be the vertex closest to u such that $u'' \in X$ and let $v'' \in P_{\mathcal{T}}$ be the vertex closest to v such that $v'' \in X$. Clearly, $dist_{\mathcal{T}}(u, v) = dist_{\mathcal{T}}(u, u'') + dist_{\mathcal{T}}(u'', v'') + dist_{\mathcal{T}}(v'', v)$. Therefore, we have $dist_{\mathcal{T}}(u, v) = dist_G(u, v) + [dist_{\mathcal{T}}(u, u'') - dist_G(u, u')] + [dist_{\mathcal{T}}(u'', v'') - dist_G(u', v')] + [dist_{\mathcal{T}}(v'', v) - dist_G(v', v)]$.

We observe that, by construction of \mathcal{T} , $dist_{\mathcal{T}}(v'', u'') \leq diam_t(X)$, $dist_{\mathcal{T}}(u, u'') \leq dist_G(u, u')$, and $dist_{\mathcal{T}}(v'', v) \leq dist_G(v', v)$. Thus, we immediately conclude that $dist_{\mathcal{T}}(u, v) \leq dist_G(u, v) + \Delta$ where $\Delta \leq diam_t(X)$. \square

Theorem 1. Let G be a graph with n vertices and m edges, (R, D) be a routing scheme with $R \leq 2D$ and $D \leq 2R$. Then G has a routing scheme with $\Delta \leq 2R$ and $O(\log^3 n / \log \log n)$ bits per edge. \square

We associate a tree $\mathcal{T}(X)$, constructed as described above, with each node X of the hierarchical tree H . Each vertex u of G only stores routing information for trees $\mathcal{T}(X)$ such that $B(u)$ is a descendant of X . Since the height of H is at most $\log n$, there are at most $\log n$ such trees. For every pair of vertices u and v we can find $X = nca_H(B(u), B(v))$. This can be done in constant time by introducing a binary label of $O(\log n)$ bits in the address of each vertex [10]. By Lemma 7, we have $dist_{\mathcal{T}}(u, v) \leq dist_G(u, v) + \Delta$, where $\Delta \leq diam_t(X) \leq 2R$.

To implement the routing in the tree $\mathcal{T}(X)$ we use the scheme presented in [7]. This scheme uses addresses and labels of length $O(\log^2 n / \log \log n)$ bits and runs in constant time.

Along the route between u and v in $\mathcal{T}(X)$, there might be vertices w such that $B(w)$ is not a descendant of X in H and therefore w does not have the routing label for the tree $\mathcal{T}(X)$. By Lemma 6, the number of such vertices is constant. We store in advance port numbers for such vertices in routing labels, which requires each vertex u to have an additional $O(\log n)$ -bit label for each of $\log n$ trees. \square

Corollary 2. Let $G = (V, E)$ be a graph with $(R, D) \leq (2, 2)$. Then G admits a routing scheme with deviation $\Delta = 2$, addresses of size $O(\log^3 n / \log \log n)$ bits per vertex and $O(1)$ routing protocol.

The proofs of the lemmas from the following sections are omitted and can be found in the full version of the paper [6].

4 Chordal-Bipartite Graphs and Interval Graphs

A bipartite graph is chordal if it does not contain any induced cycles of length greater than 4 [12]. Let $G = (X \cup Y, E)$ be a chordal bipartite graph. We construct a graph $G^+ = (X \cup Y, E^+)$ by adding edges between any two vertices $x_1, x_2 \in X$ for which there exists a vertex $y \in Y$ such that $x_1y, x_2y \in E$.

Lemma 8. Let $G = (V, E)$ be a chordal bipartite graph with $(R, D) \leq (1, 2)$. Let $\mathcal{C} = \{C_1, C_2, \dots, C_{|Y|}\}$ be a set of $|Y|$ cliques in G such that $C_i = N_G[y_i]$, $y_i \in Y$.

Theorem 2. Let $G = (V, E)$ be a graph with $(R, D) \leq (1, 2)$. Then G admits a routing scheme with deviation $\Delta = 2$, addresses of size $O(\log^3 n / \log \log n)$ bits per vertex and $O(1)$ routing protocol.

A graph G is an interval graph if it is the intersection graph of a finite set of intervals (line segments) on a line. It is well known [12] that interval graphs form a proper subclass of chordal graphs. Hence, we have

Lemma 9. Let $G = (V, E)$ be an interval graph. Then $(R, D) \leq (1, 1)$.

This lemma and Theorem 1 already imply for interval graphs existence of a loop-free routing scheme of deviation $\Delta = 2$ with addresses and routing labels of size $O(\log^3 n / \log \log n)$ bits per vertex and $O(1)$ routing protocol. In what follows, we show that even a deviation $\Delta = 1$ can be achieved.

Lemma 10. Let X be a clique in a graph $G = (V, E)$. Let $x_l, x_r \in X$ be two vertices in X . Then for any vertex $v \in V \setminus X$, $dist_G(v, X) = dist_G(v, x_l) = dist_G(v, x_r)$.

Let H be a hierarchical tree for G . For any node X of H , we construct a spanning tree \mathcal{T} of G_X in the following way. Let U be a set of vertices of G such that $U \subseteq \{V \setminus X\}$ and $B(u)$ is a descendant of X in H for any $u \in U$. For each $u \in U$, we construct a Q -simple shortest path $P(u) = \{u, z_1, \dots, z_k, x\}$ from u to X such that x is either x_l or x_r . Since X is a clique, a spanning tree $t(X)$ is a star with center at x_l . Finally, we build a graph $G_X = \bigcup_{u \in U} P(u) \cup t(X)$ and construct in a Breadth-First-Search manner starting from $t(X)$ a special spanning tree \mathcal{T} of G_X (see Lemma 5).

Lemma 11. Let $u, v \in V$ be vertices of a graph G , $X = nca_H(B(u), B(v))$, \mathcal{T} be a tree decomposition of X with $\Delta = 1$. Then $dist_{\mathcal{T}}(u, v) \leq dist_G(u, v) + \Delta$.

Theorem 3. Let G be a graph with $\Delta = 1$. Then G has a routing scheme with $O(log^3 n / log log n)$ bits per vertex and $O(1)$ routing protocol.

5 Homogeneously Orderable Graphs

A nonempty set $U \subseteq V$ is homogeneous in $G = (V, E)$ if all vertices of U have the same neighborhood in $V \setminus U$. The k -neighborhood of radius k centered at v is the set of vertices of distance at most k from v : $D(v, k) = \{u \in V : dist_G(u, v) \leq k\} = \bigcup_{i=0}^k N^i(v)$. For $U \subseteq V$ we define $D(U, k) = \bigcup_{u \in U} D(u, k)$. The k th power G^k of a graph $G = (V, E)$ is the graph with vertex set V and edges between vertices u and v with distance $dist_G(u, v) \leq k$. A subset U of V is a k -clique of G if U induces a clique in G^k . A vertex v of G with $|V| > 1$ is h -extremal if there is a proper subset $H \subset D(v, 2)$ which is homogeneous in G and for which $D(v, 2) \subseteq D(H, 1)$ holds. A vertex ordering v_1, \dots, v_n is a homogeneous elimination ordering of vertices of G if for every i , v_i is h -extremal in the induced subgraph $G_i = G(v_1 \dots v_n)$. G is homogeneously orderable if it has a homogeneous elimination ordering. As it was shown in [1], homogeneously orderable graphs include such well known classes of graphs as distance-hereditary graphs, strongly chordal graphs, dually chordal graphs, and homogeneous graphs (for the definitions see [2]). Let U_1, U_2 be disjoint sets in V . If every vertex of U_1 is adjacent to every vertex of U_2 then U_1 and U_2 form a join, denoted by $U_1 \bowtie U_2$. A set $U \subseteq V$ is a join of two non-empty sets, i.e. $U = U_1 \bowtie U_2$. The following theorem represents a well-known characterization of homogeneously orderable graphs.

Theorem 4. A graph G is homogeneously orderable if and only if G^2 is a join of two non-empty sets.

Taking into account Lemma 1 and considering $G^+ = G^2$, we conclude.

Corollary 3. Let G be a graph with $R = 2$ and $D = 2$. Then G is homogeneously orderable if and only if G is a join of two non-empty sets.

This corollary and Theorem 1 already imply for homogeneously orderable graphs existence of a routing scheme of deviation $\Delta = 4$ with addresses and routing labels of size $O(log^3 n / log log n)$ bits per vertex and $O(1)$ routing protocol. In what follows, we show that, in fact, the scheme described in Section 3 gives for homogeneously orderable graphs a routing scheme of deviation $\Delta = 2$.

Let T be a tree decomposition of a homogeneously orderable graph $G = (V, E)$ and H be its hierarchical tree. With each node $X = U_1 \bowtie U_2$ of H we associate a spanning tree \mathcal{T} of G_X as described in Section 3, where a spanning tree $t(X)$ of X is constructed as follows. Beginning at an arbitrary vertex $s_1 \in U_1$,

we visit all vertices in U_2 , then continuing from any vertex $s_2 \in U_2$, we visit all vertices in $U_1 \setminus \{s_1\}$. Clearly, $\text{diam}_t(X) = 3$.

Lemma 12.

$G, X = \text{nca}_H(B(u), B(v)), \mathcal{T} \quad X$
 $\text{dist}_{\mathcal{T}}(u, v) \leq \text{dist}_G(u, v) + \Delta \quad \Delta = 2$

Theorem 5.

$G \quad \Delta = 2 \quad O(\log^3 n / \log \log n) \quad O(1)$

The authors would like to thank Cyril Gavoille for careful reading the preliminary draft of this paper and pointing out an error in that version. As he has shown, our old method - without considering special (Q -simple) paths - could result in a scheme with labels of size $O(\log^3 n)$ bits per vertex and $O(\log \log n)$ routing protocol even on chordal bipartite graphs (although with deviation 2, not 4 as in [3]). We are grateful to him also for communicating to us the results of [3].

References

1. A. Brandstädt, F.F. Dragan, and F. Nicolai, Homogeneously orderable graphs, *Theoretical Computer Science*, 172 (1997) 209–232.
2. A. Brandstädt, V.B. Le, J. Spinrad, Graph Classes: A Survey, *SIAM Monographs on Discrete Math. Appl.*, (SIAM, Philadelphia, 1999)
3. Y. Dourisboure, Routage compact et longueur arborescente, December 2003, *PhD Thesis*, LaBRI, University of Bordeaux I.
4. Y. Dourisboure and C. Gavoille, Improved Compact Routing Scheme for Chordal Graphs, In *Proc. of the 16th Intern. Conf. on Distr. Comp. (DISC 2002)*, Toulouse, France, October 2002, *Lecture Notes in Computer Science* 2508, Springer, 252–264.
5. F.F. Dragan and I. Lomonosov, New Routing Schemes for Interval Graphs, Circular-Arc Graphs, and Permutation Graphs, In *Proc. of the 14th IASTED Intern. Conf. on Paral. and Distr. Comp. and Syst.*, Cambridge, USA, 2003, 78–83.
6. F.F. Dragan and I. Lomonosov, On Compact and Efficient Routing in Certain Graph Classes, TechReport TR-KSU-CS-2004-03, CS Dept., Kent State University, <http://www.cs.kent.edu/~dragan/TR-KSU-CS-2004-03.pdf>
7. P. Fraigniaud and C. Gavoille, Routing in trees, In *Proc. of the 28th Intern. Colloq. on Automata, Languages and Program. (ICALP 2001)*, *Lecture Notes in Computer Science* 2076, 757–772.
8. C. Gavoille, Routing in distributed networks: Overview and open problems, *ACM SIGACT News - Distributed Computing Column*, 32 (2001).
9. C. Gavoille and M. Gengler, Space-efficiency of routing schemes of stretch factor three, *Journal of Parallel and Distributed Computing*, 61 (2001), 679–687.
10. C. Gavoille, M. Katz, N. Katz, C. Paul, and D. Peleg, Approximate distance labeling schemes, *Research Report RR-1250-00*, LaBRI, University of Bordeaux, December 2000.
11. C. Gavoille and S. Pérennès, Memory requirements for routing in distributed networks, In *Proc. of the 15th Annual ACM Symp. on Principles of Distr. Comp.*, Philadelphia, Pennsylvania, 1996, pp. 125–133.

12. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
13. J. van Leeuwen and R.B. Tan, Interval routing, *The Computer Journal*, 30 (1987), 298–307.
14. D. Peleg, *Distributed computing – A locality-sensitive approach*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.
15. N. Robertson and P.D. Seymour. Graph minors. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7 (1986), 309–322.
16. N. Santoro and R. Khatib, Labeling and implicit routing in networks, *The Computer Journal*, 28 (1985), 5–8.
17. M. Thorup and U. Zwick, Compact routing schemes, In 13th *Ann. ACM Symp. on Par. Alg. and Arch.*, July 2001, pp. 1–10.
18. M. Thorup and U. Zwick, Approximate distance oracles, In 33rd *Ann. ACM Symp. on Theory of Computing (STOC)*, July 2001, pp. 183–192.

Randomized Insertion and Deletion in Point Quad Trees*

Amalia Duch

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de,
Catalunya, E-08034 Barcelona, Spain
duch@lsi.upc.es.

Abstract. In this work we introduce randomized insertion and deletion algorithms for quad trees. These algorithms are simple and they are defined for any dimension K (where K is a natural number), in contrast with the fact that standard deletion in quad trees is rather complicated and becomes more cumbersome as K increases [12]. In addition, in online applications, where long sequences of random interleaved insertions and deletions are performed, quad trees built and dynamically maintained by the randomized algorithms presented here (*randomized quad trees*) preserve their expected path length, a measure that augments when, in similar circumstances, the standard deletion algorithm is used [4]. Moreover, the expected values of random variables (such as internal path length, depth, cost of successful or unsuccessful search, cost of partial match queries, between others) given in the literature for random quad trees are valid for the randomized quad trees introduced in this work.

Keywords: Randomized Algorithms, Quad trees.

1 Introduction

The quad tree, first introduced by Finkel and Bentley [5], is a data structure widely used in several computer applications, such as geographical information systems, databases or computer graphics [13]. Such applications require the dynamic maintenance of a set (or file) of n multidimensional records. Each multidimensional record contains an ordered K -tuple of values that constitute its K -dimensional (or multidimensional) key and associated information.

Informally, a quad tree T stores multidimensional keys as follows. Given a sequence of n points in a K -dimensional space, say $[0, 1]^K$, the first point is taken as root of the quad tree. The hyper planes parallel to the axes through this point cut the space into 2^K hyper quadrants corresponding to the 2^K subtrees of T . All the points that belong to the same hyper quadrant will be stored in the same subtree. The procedure continues recursively in each hyper quadrant until there

* This research was partially supported by the Future and Emergent Technologies programme of the EU under contract IST-1999-14186 (ALCOM-FT) and the Spanish Min. of Science and Technology project TIC2002-00190 (AEDRI II).

are no remaining points. The term *point quad tree* has been coined to distinguish the tree just described from other kinds of quad trees [13]. In this work the term *quad tree* will be used to mean *point quad tree*.

Quad trees are frequently used in applications because they support a large set of operations with simple algorithms (except for deletions) and reasonable time requirements. As long as the dimension K is not too big (less or equal than 3), the space requirements are also acceptable. The efficient expected case performance of quad trees holds under the assumption that they are *random*. A quad tree is random if it is incrementally built by inserting keys drawn independently from a continuous distribution in $[0, 1]^K$. However, in dynamic and on-line settings, where long sequences of interleaved insertions and deletions must be performed over a quad tree, its total path length (a measure of the average cost of accessing a node) augments significantly, even if the quad tree was random or perfectly balanced [4]. This happens even if every node of the tree is equally likely to be deleted. After some updates (insertions and deletions), the tree may need to be rebuilt to preserve its efficient expected performance.

In this work we present one possibility to overcome dynamically the problem of poor performance by means of randomized algorithms. We draw upon the ideas of [3, 10] to randomize quad trees, in order to present update algorithms that preserve the randomness of the tree, independently of the order of previous insertions and deletions. The difference between these algorithms and the ones for binary trees depends only on the intrinsic characteristics of quad trees. The resulting update algorithms are simple and defined for any dimension K . The case of deletions is of particular interest because the classic deletion algorithm over quad trees [12] is rather complex and its description becomes more complicated as the dimension increases.

This work is organized as follows. In Section 2 we give the definition and preliminaries of quad trees, with special emphasis in insertion and deletion algorithms. In Section 3 we introduce randomized insertion and deletion algorithms and we give their properties in Section 4, where we prove that the randomized algorithms presented here always produce random quad trees.

2 Quad Trees

In this work, for the sake of simplicity, we identify a multidimensional record with its corresponding K -dimensional key $x = (x_0, x_1, \dots, x_{K-1})$, where each x_i , ($0 \leq i \leq K - 1$), refers to the value of the i -th attribute of the key x . Each x_i belongs to some totally ordered domain D_i , and x is an element of $D = D_0 \times D_1 \times \dots \times D_{K-1}$. Therefore, each (multidimensional) key may be viewed as a point in a K -dimensional space, and its i -th attribute can be viewed as the i -th coordinate of such a point. Without loss of generality, we assume that $D_i = [0, 1]$, for all $0 \leq i \leq K - 1$, and hence that D is the hypercube $[0, 1]^K$ [8].

We say that two K -dimensional keys x and y are *compatible* if, for every $i = 0, 1, \dots, K - 1$, their i -th attributes are different. Note that any two keys drawn uniformly and independently from $[0, 1]^K$ are compatible, since the probability that $x_i = y_i$ is zero.

Given a bit string $w = w_0w_1 \cdots w_{K-1}$ of length K (that is, $w \in \{0, 1\}^K$) we define the binary relation \prec_w between K -dimensional keys that are compatible as follows.

Definition 1. \prec_w is defined as follows: $x \prec_w y$ iff $i \in \{0, 1, \dots, K - 1\}$, $x_i < y_i$ and $w_i = 0$ and $x_i > y_i$ and $w_i = 1$ and $x \not\prec_w y$.

Note that for every $w \in \{0, 1\}^K$ the relation \prec_w is antisymmetric and transitive. Note also that if $x \prec_w y$, then $y \prec_{\bar{w}} x$, where \bar{w} is the complementary bit string of w in $\{0, 1\}^K$, that is, $\bar{w}_i = 0$ iff $w_i = 1$ ($i \in \{0, 1, \dots, K - 1\}$). Furthermore, $x \prec_w y$ for exactly one bit string $w \in \{0, 1\}^K$ and $x \not\prec_w y$ for all other bit strings. We are now ready to give the definition of K -dimensional quad trees that we will use in this work.

Definition 2. A quad tree T of order K is a rooted tree with 2^K leaves. The root node is labeled t_w where $w \in \{0, 1\}^K$. The root node has four children labeled x , w , y and $x \prec_w y$.

Abusing language, we will say w -th subtree or w -th quadrant referring to subtree w or quadrant w , irrespectively.

The standard insertion algorithm immediately follows from the previous definition: compare the new elements' key with the key at the root of the quad tree and obtain the index w of the subtree where the insertion must recursively proceed.

The deletion algorithm, first introduced by Samet [12], is not so straightforward. Analogous to deletion in binary search trees, the goal is to replace the deleted node with the closest possible one. But in quad trees it is not clear which of the remaining nodes is the closest in all dimensions simultaneously, and no matter which one is chosen, a reorganization of some nodes will be required. Here is the description of the algorithm for the two-dimensional case (it becomes more complex as the dimension increases): let x be the node to be deleted and y the node that is going to replace it. Ideally we would like to choose y such that the region R (see Figure 1(a)) between the orthogonal lines passing through x and y is empty. Unfortunately, this is not always possible. In a two-dimensional quad tree there are four nodes (one for each subtree of x) that are candidates to be closest to x (see [12]). So, the first step of the algorithm is to find them. For each subtree w of x , the candidate node corresponding to subtree w is found starting the search at the root node of w and systematically following the \bar{w} -th subtree until a node with empty \bar{w} -th subtree is reached (the node we are looking for). Such a node is shown in Figure 1(b). The second step of the algorithm consists of choosing from these four candidates the node y that will replace the deleted node x . This is done applying the following criteria.

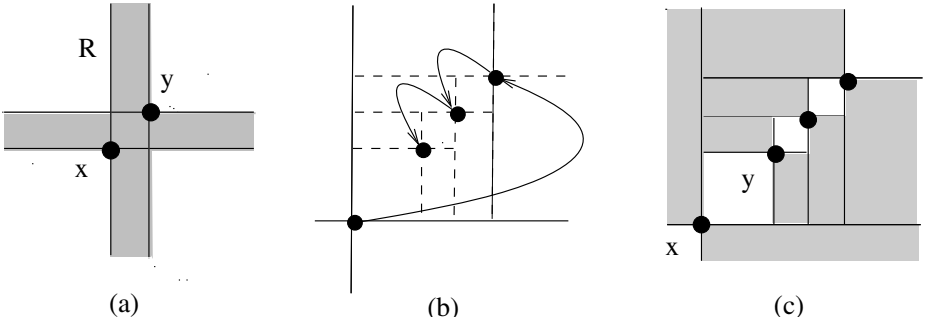


Fig. 1. Some aspects of Samet’s deletion algorithm: (a) the desired empty area R (in grey), (b) the traversal of one quadrant for finding a replacement node, (c) the processed sub-quadrants by the `adjquad` procedure (in grey), when node x is deleted and replaced by node y

- **Criterion 1.** Choose the node that is closer to each of its bordering axes than any other candidate on the same side of these axes, if such a candidate exists.
- **Criterion 2.** If no candidate satisfies Criterion 1, then the candidate with the minimum L_1 metric value (sum of the displacements from the bordering axes) should be chosen.

Once the replacement node y is chosen, the next step of the algorithm is to determine which are the nodes of the tree that require reinsertion. This is done by means of two procedures: `adjquad` (adjacent quadrant) and `new_root`.

Let w be the subtree of x (the node to be deleted) that contains y (the replacement node). Note that no node of subtree \bar{w} needs reinsertion. But the two sub-quadrants adjacent (\bar{w} is the opposite) to subtree w must be processed separately by the `adjquad` procedure (in grey in Figure 1(c)). Once the nodes in quadrants adjacent to quadrant w have been processed, it is the turn of nodes in quadrant w . Clearly all nodes in sub-quadrant w of quadrant w will retain their position. So, procedure `new_root` is applied to the remaining sub-quadrants of quadrant w .

Theoretical and empirical results for the above deletion algorithm [12] show that, comparing against the complete reconstruction of the subtree beneath the deleted node, for independently and uniformly distributed data in $[0, 1]^K$, the average number of nodes requiring reinsertion is reduced by a factor of $5/6$ (83%) when the replacement node satisfies Criteria 1 and 2. When the replacement node is chosen randomly among the four possible candidates the average number of nodes requiring reinsertion is reduced by a factor of $2/3$ (67%). This implies that this deletion algorithm has the same order of growth as a complete reconstruction of the subtree whose root is to be deleted. The expected cost is thus $\mathcal{O}(\log n \cdot \log \log n)$ for a tree of size n , since the size of the subtree beneath the deleted node is $\mathcal{O}(\log n)$ [9].

Empirical tests show that the number of comparisons required to locate the deleted node in a quad tree of n nodes is proportional to $\log n$ and that the total path length of the tree after a single deletion decreases slightly. However, as it is shown in [4], for long sequences of insertions and deletions the total path length augments and so do the deletion costs.

3 Randomized Insertions and Deletions in Quad Trees

The randomized insertion and deletion algorithms presented here require that each node stores the size of the subtree beneath it [10], because their behavior depends on the size of the (sub)trees to which they are applied.

Without loss of generality, we assume in what follows that randomized algorithms have free access to a source of random bits and that the cost of generating a random number of $\mathcal{O}(\log n)$ bits is constant [11].

Informally, we would like insertion and deletion algorithms that produce quad trees that behave as if they were built by successive insertions of uniformly and independently generated multidimensional keys.

In order to produce such quad trees it is required that any new inserted key has some probability of becoming the root of the tree, or the root of one of the subtrees of the root, and so forth. In the same vein, when a node is deleted it is required that all the remaining nodes in the subtree beneath it have some probability of replacing it. The randomized insertion and deletion algorithms that we present now provide these capabilities.

The randomized insertion algorithm of x in a tree T ($\text{insert}(T, x)$, where x is compatible with T), proceeds as follows.

1. If the tree T is the empty tree, then the algorithm insert produces a tree with root node x and 2^K empty subtrees.
2. If the tree T is not empty and has size n , then, with probability $\frac{1}{n+1}$ the key x must be placed at the root of the new tree using the insert_at_root algorithm (since the new tree will have size $n + 1$). Otherwise, we insert x recursively in the corresponding w -th subtree of T depending on \prec_w (x 's order relation with the root of T).

The algorithm insert requires the possibility of inserting the key x at the root of any subtree of a quad tree T . If T is empty, then, $\text{insert_at_root}(T, x)$ (described as Algorithm 1) gives as a result a tree with root node x , and empty subtrees. When T is not empty, $\text{insert_at_root}(T, x) = T'$ where, by definition, the root of T' is x and, its w -th subtree consists of all those elements z of T such that $z \prec_w x$. To obtain the subtrees of T' we use the split algorithm which we present later on.

We use the notation $p \rightarrow \text{field}$ to refer to the field \dots in the node pointed to by p . Usually, a quad tree is represented by a pointer to its root node and nodes have fields key , size and subtrees w (for all $w \in \{0, 1\}^K$).

The deletion of a record from a random quad tree involves searching the key x to be deleted in the tree, and then joining its corresponding 2^K subtrees. Since

Algorithm 1. The insertion at the root of randomized quad trees

```

{Initial call:  $T := \text{insert\_at\_root}(T, x)$ }
function  $\text{insert\_at\_root}(T, x)$ 
  for ( $w \in \{0, 1\}^K$ )
     $S_w := \text{split\_w}(T, x);$ 
  end
   $T \rightarrow \text{key} := x;$ 
   $T \rightarrow w := S_w;$ 
  return  $T;$ 
end

```

it is required that all the nodes in these subtrees have some probability of taking the place of the deleted node, we require the join algorithm (introduced below) which achieves this capability.

Observe that both, insertions and deletions, consist of two different steps. A first step in which one must follow a path in the tree in order to locate the place where the key must be inserted or deleted and, a second step in which the update is performed.

The expected cost of insertions and deletions in a random quad tree of n nodes is $\mathcal{O}(\log n)$, since both the insertion at the root of any element and the deletion of any element occur near the leaves on the average and the expected size of a random subtree in a random quad tree of size n is $\mathcal{O}(\log n)$. As we will see below the cost of the required operations is linear with respect to the size of the trees to which they are applied, hence the claimed cost follows. In contrast, the average cost of Samet's deletion algorithm is $\mathcal{O}(\log n \log \log n)$ when it is applied to trees of size $\mathcal{O}(\log n)$. Unfortunately we only have empirical evidence of this fact, because a theoretical approach has proven to be extremely difficult. The reason is that, contrary to the one-dimensional case, the analysis implies the solution of a non-linear system of recurrences derived from interleaved randomized operations.

The insertion of a key x at the root of a tree T (the task that performs $\text{insert_at_root}(T, x)$) is accomplished in two steps: a first step in which the tree T is partitioned with respect to x producing the 2^K trees T'_u , for all $u \in \{0, 1\}^K$, where T'_u contains all those keys z of T such that $z \prec_u x$; and a second step in which the 2^K subtrees are attached to the new root x . Clearly the main cost of $\text{insert_at_root}(T, x)$ lies in the partitioning or splitting process $\text{split}(T, x)$.

To simplify the description of the split algorithm we will see it as a 2^K -tuple of functions split_u , for each $u \in \{0, 1\}^K$, with $T'_u = \text{split}_u(T, x)$. In practice all the T'_u trees can be simultaneously computed. This algorithm is depicted by Algorithm 2.

The algorithm split_u works in the following way. When T is the empty tree, the split_u algorithm returns the empty tree. Otherwise, let T have root y and subtrees T_w for all $w \in \{0, 1\}^K$. For each $u \in \{0, 1\}^K$ we have the following cases to consider.

1. If $y \prec_w x$ and $w = u$, then y belongs to T'_u , all the elements of T_w do as well (because \prec_w is transitive), and the operation proceeds recursively in the rest of subtrees in order to complete the result;
2. If $y \prec_w x$ and $w \neq u$, then y does not belong to T'_u , neither do all the elements of T_w (because \prec_w is transitive), and the operation proceeds recursively in the remaining subtrees producing $2^K - 1$ subtrees that must be brought together by means of the join algorithm;

It is not difficult to see that all the $\text{split}_u(T, x)$ compare x against the keys of $2^K - 1$ subtrees of T while the split algorithm compares x against all the keys of T . The additional cost of the join algorithm must be also taken into account. So, the split algorithm has running time at least linear, but smaller than a complete reconstruction of a (sub)tree. Moreover, in the average, it is applied to relatively small subtrees, i.e., of size $\mathcal{O}(\log n)$ [9].

Algorithm 2. The split_u algorithm for randomized quad trees

```

{Initial call:  $T'_u := \text{split}_u(T, x)$ }
function  $\text{split}_u(T, x)$ 
   $y := T \rightarrow \text{key};$ 
  for ( $w \in \{0, 1\}^K$ )
    if ( $y \prec_w x$ ) then
      if ( $w = u$ ) then
         $T'_u \rightarrow \text{key} := y;$ 
         $T'_u \rightarrow w := T_w;$ 
        for ( $v \in \{0, 1\}^K, v \neq w$ )
           $T'_u \rightarrow v := \text{split}_u(T \rightarrow v, x);$ 
        end
      else
         $T'_u := \text{join}(\text{split}_u(T \rightarrow 00\dots 0, x), \dots, \text{split}_u(T \rightarrow 11\dots 1, x));$ 
      fi
    fi
  end
  return  $T'_u;$ 
end

```

We now describe the algorithm $\text{join}(T_{w_0}, T_{w_1}, \dots, T_{w_{2^K-1}})$ depicted in Algorithm 3. By definition this algorithm is applied only when the keys in the trees T_w are related to a key y by relations \prec_w respectively.

As we have already pointed out, in order to produce random quad trees, each node of the trees $T_{w_0}, T_{w_1}, \dots, T_{w_{2^K-1}}$ must have some probability of becoming the root of the new tree.

If $T_{w_0}, T_{w_1}, \dots, T_{w_{2^K-1}}$ have sizes $n_0, n_1, \dots, n_{2^K-1}$ respectively, then, $T = \text{join}(T_{w_0}, T_{w_1}, \dots, T_{w_{2^K-1}})$ has size $n = n_0 + n_1 + \dots + n_{2^K-1}$. Thus, the join algorithm selects, with probability $\frac{n_i}{n}$ the root of T_{w_i} as root of T . We have the following.

1. If the trees $T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}}$ are all empty, then the result of joining them, $T = \text{join}(T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}})$, is the empty tree;
2. If at least one of them is not empty, then let $T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}}$ have roots $y_0, y_1, \dots, y_{2^k-1}$ and subtrees $t_{u_j}^0, t_{u_j}^1, \dots, t_{u_j}^{2^k-1}$ ($j = 0, \dots, 2^k - 1$), respectively. In this situation, if the tree T has root node y_i , we have the following.
 - (a) All the keys of the tree $t_{u_j}^i$ should be members of the w_j -th subtree of T for all $j \in \{0, 1, \dots, 2^k - 1\}$.
 - (b) The whole tree $T_{\bar{w}_i}$ becomes part of the \bar{w}_i -th subtree of T , since, for every $z \in T_{\bar{w}_i}$, $z \prec_{\bar{w}_i} y \prec_{\bar{w}_i} y_i$ (where y is the deleted node).
 - (c) The $2^k - 2$ remaining trees must be split with respect to y_i and the corresponding trees joined together.

Algorithm 3. The join algorithm for randomized quad trees

```

{Initial call:  $T := \text{join}(T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}})$ }
function  $\text{join}(T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}})$ 
  for ( $i = 0, \dots, 2^k - 1$ )
     $y_i := T_{w_i} \rightarrow \text{key}; n_i := T_{w_i} \rightarrow \text{size};$ 
    for ( $j = 0, \dots, 2^k - 1$ )
       $t_{u_j}^i = T_{w_i} \rightarrow u_j;$ 
    end
  end
   $\text{total} := n_0 + n_1 + \dots + n_{2^k-1}; n_{-1} := 0; n_{2^k} := \text{total};$ 
   $r := \text{rand}(0, \text{total} - 1);$ 
  for ( $i = 0, \dots, 2^k - 1$ )
    if ( $n_i \leq r < n_{i+1}$ ) then
       $T \rightarrow \text{key} := y_i;$ 
      for ( $j = 0, \dots, 2^{k-1} - 1$ )
         $S_{w_0} := \text{split\_}\{\mathbf{w\_j}\}(T_{w_0}, y_i);$ 
         $S_{w_1} := \text{split\_}\{\mathbf{w\_j}\}(T_{w_1}, y_i);$ 
         $\vdots$ 
         $S_{w_{2^k-1}} := \text{split\_}\{\mathbf{w\_j}\}(T_{w_{2^k-1}}, y_i);$ 
         $T \rightarrow w_j := \text{join}(S_{w_0}, \dots, t_{u_j}^i, \dots, S_{w_{2^k-1}});$ 
      end
    fi
  end
  return  $T;$ 
end

```

Observe that, the join algorithm traverses the tree in a similar way than the split_{u_i} algorithm, since the join algorithm is continued recursively in $2^k - 1$ subtrees. The additional cost of the split algorithm in $2^k - 2$ subtrees should

also be taken into account. But once again, since the join algorithm is applied near the leaves in the average, its expected cost is $\mathcal{O}(\log n)$ [9].

We say that a quad tree is a *randomized quad tree* if it is the result of a sequence of update operations performed by means of the randomized algorithms introduced below, applied to an initially empty tree. We shall show in the next section that any tree obtained this way is a random quad tree.

4 Properties of Randomized Quad Trees

In this section we prove that the result of any arbitrary sequence of randomized insertions and deletions starting from an initially empty quad tree is always a randomized quad tree. This result implies that the theoretical results given in the literature for random quad trees hold in practice when the randomized algorithms presented here are used.

We have previously mentioned that a quad tree of size n is *random* if it is built by n insertions of multidimensional keys independently drawn from a continuous distribution in $[0, 1]^K$ (for simplicity let us assume a uniform distribution). This assumption about the distribution of the input implies that the $n!^K$ distinct configurations of input sequences are equally likely. In particular, in a random quad tree of size n , each of the n possible K -dimensional keys are equally likely to appear in the root, and once the root is fixed, the 2^K subtrees are independent random quad trees.

Observe that, unlike binary search trees, the external nodes of a K -dimensional quad tree are not equally likely to be the position of the next insertion. However, observe that, for $n > 2$, in an input sequence of n K -dimensional keys, the last key can not be in the root of a quad tree, it must belong to one of the 2^K subtrees. Given n_j , the size of the j -th subtree after n insertions ($j = 0, 1, \dots, 2^K$), any of the n_j keys could be the last one and thus the last key is in the j -th subtree with probability $n_j/(n - 1)$ [8].

The randomized split and join algorithms preserve the randomness of their input (Lemma 1 below). In other words, when applied to random quad trees, both the split and the join algorithms produce random quad trees. Moreover, since this happens, the insert and delete algorithms when applied to random quad trees produce random quad trees. These claims are formalized as follows.

Lemma 1. *i)* $T = \text{split}(T, x) \rightsquigarrow T_{u_i} \dots T_{u_{2^K}}$ ($u_i \in \{0, 1\}^K$)
ii) $T' = \text{join}(T_{w_0}, T_{w_1}, \dots, T_{w_{2^K-1}}) \rightsquigarrow T_{w_i} \dots T_{w_{2^K-1}}$
 $T_{w_i} \rightsquigarrow T_{w_i} \dots T_{w_{2^K-1}}$

We prove the two parts of this lemma by induction on the size n of T to show that split preserve randomness, and on the joint size $n = |T_{w_0}| + |T_{w_1}| + \dots + |T_{w_{2^K-1}}|$ of T' to show that join also preserves randomness. Observe that to prove the two parts of the lemma for size n , we will need to inductively and

simultaneously assume that both statements are true if T (T') is of size smaller than n . The reason for that is the mutual recursion between the split_u operations and join .

If T is empty ($n = 0$) then $\text{split}_u(T, x)$ is an empty tree for all $u \in \{0, 1\}^K$, and hence the first part of the lemma trivially holds for the basis of the induction. Also, if $T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}}$ are empty ($n = |T_{w_0}| + |T_{w_1}| + \dots + |T_{w_{2^k-1}}| = 0$) then T' is empty and hence random.

Let us consider the case where $n > 0$, assuming now that both parts of the lemma are true for all sizes smaller than n .

We start with the split process, and let n denote the size of T , the quad tree to be partitioned. Let the root of T be y and let $t_{u_0}, t_{u_1}, \dots, t_{u_{2^k-1}}$ denote its subtrees.

If $y \prec_{u_i} x$, then y is the root of the tree T_{u_i} and t_{u_i} its u_i -th subtree. To complete T_{u_i} we apply recursively the split algorithm to the subtrees $t_{u_0}, t_{u_1}, \dots, t_{u_{i-1}}, t_{u_{i+1}}, \dots, t_{u_{2^k-1}}$. Since $|t_{u_i}| < n$, by the inductive hypothesis, we obtain the $2^k - 1$ random and independent subtrees of T_{u_i} . Moreover, these subtrees are independent of t_{u_i} since they are obtained from the subtrees $t_{u_0}, t_{u_1}, \dots, t_{u_{i-1}}, t_{u_{i+1}}, \dots, t_{u_{2^k-1}}$, which, by hypothesis are independent of t_{u_i} . Simultaneously, from subtrees $t_{u_0}, t_{u_1}, \dots, t_{u_{i-1}}, t_{u_{i+1}}, \dots, t_{u_{2^k-1}}$, we obtain $2^k - 1$ trees for each subtree $T_{u_0}, T_{u_1}, \dots, T_{u_{i-1}}, T_{u_{i+1}}, \dots, T_{u_{2^k-1}}$. These trees are of size smaller than n and by inductive hypothesis, they are random and independent, so, also by inductive hypothesis, the result of joining them is a random quad tree, and thus $T_{u_0}, T_{u_1}, \dots, T_{u_{2^k-1}}$ are all random and independent. To complete the proof for this case, we need only show that for every key \mathbf{z} of T_{u_i} the probability that \mathbf{z} is at the root of T_{u_i} is $\frac{1}{m}$, where m is the size of T_{u_i} . Indeed,

$$\begin{aligned} \mathbb{P}\{\{\mathbf{z} \text{ is root of } T_{u_i} \mid y \prec_{u_i} x\}\} &= \frac{\mathbb{P}\{\{\mathbf{z} \text{ is root of } T \text{ and } \mathbf{z} \prec_{u_i} x\}\}}{\mathbb{P}\{\{\mathbf{z} \prec_{u_i} x\}\}} \\ &= \frac{1/n}{m/n} = \frac{1}{m}. \end{aligned}$$

Now we tackle the second part of the lemma and show that join preserves randomness when $n > 0$. If exactly one of the trees $T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}}$ is not empty, then join returns the non-empty tree which, by hypothesis, is random. We consider now the case where at least two of the trees $T_{w_0}, T_{w_1}, \dots, T_{w_{2^k-1}}$ are not empty. Let T_{w_i} have root y_i , and subtrees $t_{w_j}^i$, (for all $j \in \{0, 1, \dots, 2^k - 1\}$). If we select the key y_i to become the root of T' , then we will recursively join $t_{w_j}^i$ with the corresponding trees that result from splitting the trees $T_{w_0}, T_{w_1}, \dots, T_{w_{i-1}}, T_{w_{i+1}}, \dots, T_{w_{2^k-1}}$ with respect to y_i . By the inductive hypothesis, from this splitting process result independent and random trees of sizes smaller than n . Thus, again by the inductive hypothesis joining some of them together produces a random quad tree. Thus, we have that the w_i -th subtree of T' is $t_{w_i}^i$, which is a random quad tree and that the other subtrees are also random since they are the result of splitting and joining subtrees of size smaller than n . All the subtrees are independent of each other and the probability that y_i was the root of T_{w_i} times the probability that it is selected as root of T' is $\frac{1}{|T_{w_i}|} \times \frac{|T_{w_i}|}{n} = \frac{1}{n}$. \square

Theorem 1. $T \xrightarrow{\text{insert}(T, x)} X$
 $X \xrightarrow{\text{insert}(T, x)} X \cup \{x\}$

By induction on the size n of T . If $n = 0$, then T is the empty tree (a random quad tree), and $\text{insert}(T, x)$ returns a random quad tree with root x , and 2^K empty subtrees. We assume now that T is not empty and that the theorem is true for all sizes smaller than n . The insertion of x in T has two possible results, with probability $\frac{1}{(n+1)}$, x is the root of $T' = \text{insert}(T, x)$ and with complementary probability x is recursively inserted in the corresponding w_i -th subtree of T .

Let us consider first the case in which x is not inserted at the root of T . Consider an record $y \in T$. The probability that y is the root of T before the insertion of x is $\frac{1}{n}$, since by hypothesis T is a random quad tree. The probability that y is at the root of T' is the probability that x is not at the root of T' and y was at the root of T , which is $\frac{1}{n} \times \frac{n}{n+1}$, resulting in the desired probability. Moreover, since x is not inserted at the root of T' in the first step, the insertion proceeds recursively in one of the subtrees of T , which are independent random quad trees of sizes smaller than n . Thus, T' is a random quad tree of size $n + 1$.

Finally, with probability $\frac{1}{n+1}$, x is the root of T' . The tree T , which by hypothesis is a random quad tree, must be split with respect to x , and because of Lemma 1 this step produces 2^K independent random quad trees which are the subtrees of T' , thus T' is also a random quad tree. \square

Theorem 2. $T \xrightarrow{\text{delete}(T, x)} T'$
 $T' \xrightarrow{\text{delete}(T, x)} X \setminus \{x\}$

If the key x is not in T , then the algorithm does not modify the tree, which is random. Let us now suppose that x is in T , this case is proved by induction on the size of the tree. If $n = 1$, x is the only key in the tree and after deletion we obtain the empty tree which is a random quad tree. We now assume that $n > 1$ and that the theorem is true for all sizes smaller than n . If x was not the key at the root of T we proceed recursively in one of the 2^K subtrees, and by inductive hypothesis we obtain a randomly built subtree. If x was the key at the root of T , the tree after deletion is the result of joining the 2^K subtrees of T , which produce a random quad tree because of Lemma 1. Finally, after deletion, each node has probability $\frac{1}{(n-1)}$ of being the root. Let y be any key in T such that $y \neq x$, then

$$\begin{aligned} \mathbb{P}[\{y \text{ is the root of } T'\}] &= \mathbb{P}[\{y \text{ is the root of } T' \mid x \text{ was not the root of } T\}] \\ &\quad \times \mathbb{P}[\{x \text{ was not the root of } T\}] \\ &\quad + \mathbb{P}[\{y \text{ is the root of } T' \mid x \text{ was the root of } T\}] \\ &\quad \times \mathbb{P}[\{x \text{ was the root of } T\}] \\ &= \frac{1}{n-1} \times \frac{n-1}{n} + \frac{1}{n-1} \times \frac{1}{n} \\ &= \frac{1}{n-1} \end{aligned}$$

Thus, we obtain the desired probability. \square

Combining the two previous theorems we obtain the following important corollary.

Corollary 1.

Several random variables over random quad trees, which hold also for randomized quad trees, have been studied in the literature. For instance, the expected depth of insertion D_n of the n -th multidimensional key is in probability asymptotic to $(2/K) \log n$ [2]. The expected height H_n of a K -dimensional quad tree of size n is in probability asymptotic to $(c/K) \log n$, where $c = 4.31107\dots$ [1]. The cost of a random search has logarithmic mean and variance and is asymptotically distributed as a normal variable [7]. The cost of a partial match query with s coordinates specified is $\Theta(n^{1-s/K+\theta(s/K)})$, where the function $\theta(x)$ is defined as the solution $\theta \in [0, 1]$ of the equation $(\theta + 3 - x)^x(\theta + 2 - x)^{1-x} - 2 = 0$ [6].

The deletion algorithm here presented is not only asymptotically more efficient than the standard one, but it is also simpler and scales smoothly for larger dimensions.

References

1. Devroye, L., Branching processes in the analysis of the height of trees. *Acta Informatica*, 24:277-298, 1987.
2. Devroye, L. and Lafortest, L., An Analysis of Random d -Dimensional Quadrees. *SIAM Journal of Computing*, 19(5):821-832, 1990.
3. Duch, A., Estivill-Castro V. and Martínez, C., Randomized K -Dimensional Binary Search Trees. *Algorithms and Computation 9th International Symposium, ISAAC'98, Taejon, Korea*, LNCS(1533):199-208, 1998.
4. Eppinger, J. L., An empirical study of insertion and deletion in binary search trees. *Communications of the ACM*, 26(9):663-669, 1983.
5. Finkel, R. A. and Bentley, J. L., Quadrees, a Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4:1-9, 1974.
6. Flajolet, Ph., Gonnet, G., Puech, C. and Robson, J. C., Analytic Variations on Quadrees. *Algorithmica*, 10:473-500, 1993.
7. Flajolet, Ph. and Lafforgue, T., Search Costs in Quadrees and Singularity Perturbation Analysis. *Discrete and Computational Geometry*, 12(4):151-175, 1993.
8. Mahmoud, H. M., *Evolution of Random Search Trees*. Wiley Interscience Series, 1992.
9. Martínez, C., Panholzer, A. and Prodinger, H., On the number of descendants and ascendants in random search trees. *Electronic Journal on Combinatorics*, 5(1), 1998.
10. Martínez, C. and Roura, S., Randomized binary search trees. *Journal of the ACM*, 45(2):288-323, 1998.
11. Motwani, R. and Raghavan, P., *Randomized Algorithms*. Cambridge University Press, Cambridge, USA, 1995.
12. Samet, H., Deletion in Two-Dimensional Quadrees. *Communications of the ACM*, 23(12):703-710, 1980.
13. Samet, H., *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

Diagnosis in the Presence of Intermittent Faults*

Bin Fu^{1,2} and Richard Beigel³

¹ Computer Science Department,
University of New Orleans, New Orleans, LA 70148, USA
`fu@cs.uno.edu`

² Research Institute for Children,
200 Henry Clay Avenue, New Orleans, LA 70118, USA

³ Dept. of Computer and Information Sciences,
Temple University, Philadelphia, PA 19122, USA
`beigel@cis.temple.edu`

Abstract. Several constant-time parallel algorithms for fault diagnosis have been given for the classical static-fault model. We extend those algorithms to tolerate intermittent faults.

1 Static Fault Diagnosis

Imagine a robot surveying a nuclear accident. As it collects data in radioactive hot spots, its processors tend to get fried. Before too many processors fail, the robot retreats to a safe spot in order to heal. During the healing process, the robot's processors test each other and communicate test results (some correct, some incorrect) to an external controller in a very safe location as far as possible from the accident. The controller determines which processors are really faulty and sends instructions to deactivate them. Then the robot may return to its mission.

The robot's "healing" process is called system level fault diagnosis, as proposed initially by Preparata, Metzger, and Chien [9]. In their model, when a good processor tests another processor it reports the testee's status correctly to the controller. But when a faulty processor tests another processor it may report the testee's status correctly or incorrectly. Such testing protocols have been usefully implemented, see [4, 3].

Fault diagnosis is not possible in general unless a strict majority of the processors are good [9]. We will assume that throughout this paper.

In this paper, we consider fault diagnosis (proposed by Nakajima [8] as an extension to Preparata et al's model), we assume a complete interconnect like an ethernet, so any processor can test any other. The testing process is divided into rounds, during which exactly one test may be performed. The controller dictates which

* Research is supported from the National Science Foundation under grants CCR-9796317 and CCR-9700417, and Louisiana Board Regent under grant LEQSF(2004-07)-RD-A-35.

test is performed in each round, and then receives the result of that test. After a finite number of rounds, the controller must determine which processors are really faulty. We assume that the fault status of processors is constant throughout the testing (in our example the robot remains in a safe haven throughout the testing process). The controller is, of course, assumed to be reliable as well.

This led naturally to *adaptive parallel fault diagnosis*, where each processor may participate in at most one test per round. Adaptive parallel fault diagnosis can be performed in 10 rounds [1].

A related problem, posed by Manber [7], is to identify one faulty processor, rather than all of them. This is called “diagnosis with repair” because the faulty processor can be replaced and then the process can be repeated with one fault fewer.

2 Dynamic Faults and Errors

Preparata et al’s model fails to address at least two practical issues:

- (1) processors may fail during the diagnosis procedure
- (2) faults may be intermittent

Hurwood [5] addressed the first of these issues by extending Manber’s diagnosis-with-repair model. In his model, all processors are initially good; at most t processors can fail per round and at most t processors can be repaired per round. Hurwood gives an algorithm that maintains indefinitely a dynamic equilibrium in which the number of faulty processors is $O(t \log t)$.

A processor is intermittent if the test result is arbitrary whenever it is involved in a test – either as tester or testee. Formally we allow three distinct kinds of processors: good, faulty, and intermittent. A good processor always gives a correct answer when it tests a good processor or a faulty processor, but gives an arbitrary answer when it tests an intermittent processor. A faulty processor gives an arbitrary answer when it tests any processor. An intermittent processor also gives an arbitrary answer when it tests any processor. Throughout this paper, we consider a set of processors P . We will let $\text{good}(P)$ denote the set of good processors in P , $\text{intermittent}(P)$ the set of intermittent processors in P , and $\text{faulty}(P)$ the set of faulty processors in P . For a set A , $|A|$ denotes the number of elements in it. We will assume that $|\text{good}(P)| > |\text{faulty}(P)| + |\text{intermittent}(P)|$. Note that $\text{good}(P)$, $\text{faulty}(P)$, and $\text{intermittent}(P)$ are pairwise disjoint sets.

In our model, we cannot guarantee to diagnose intermittent processors, because they might, for example, behave as good in every test. While intermittent processors may be misdiagnosed, we guarantee to correctly diagnose all consistently faulty processors and all good processors.

. Intermittent Diagnosis Assuming that $|\text{intermittent}(P)| \leq t$, find a set S such that $\text{faulty}(P) \subseteq S \subseteq \text{faulty}(P) \cup \text{intermittent}(P)$.

We give an $O(t^5)$ -rounds algorithm for t -Intermittent Diagnosis.

3 Fault Diagnosis with a Bounded Number of Intermittent Processors

In this section we permit three kinds of processors: good, faulty, and intermittent. An intermittent processor may act good or faulty at any time. It makes the situation more complicated. We describe an algorithm for diagnosis assuming that a strict majority of the processors are good and that there are no more than t intermittent processors.

Definition 1

- Let p, q be two processors. p *likes* q if the result is good when p tests q . p *dislikes* q if both results are good when p tests q and q tests p .
- A *super-node* is a set of 2^k processors for some number $k \geq 0$. For two super-nodes A, B , $\langle A, B \rangle$ is a *pair*. For a pair (A, B) , $\text{partner}(A) = B$ and $\text{partner}(B) = A$.
- Let S be a set of equal-size super-nodes and $G' \subseteq S$. If $|G'| \geq \frac{1}{6}|S|$, G' is called a *majority node* of S .
- For two sets of processors A and B , a *k-test* from A to B takes k processors p_1, \dots, p_k from A and k processors q_1, \dots, q_k from B and p_i and q_i test each other for $i = 1, \dots, k$. The result is called (k, t) -*good* if at least t of the k pairs like each other.
- For two sets of processors A and B , a *k-test* from A to B is to select k processors p_1, \dots, p_k from A and k processors q_1, \dots, q_k from B and let p_i test q_i . The result is (k, t) -*good* if at least t of the k tests are good.
- For a finite set A and an integer m , an *m-partition* is a sequence of sets A_1, \dots, A_k such that $|A_1| = \dots = |A_{k-1}| = m$, $|A_k| \leq m$, $A_i \cap A_j = \emptyset$ for $i \neq j$, and $\bigcup_{i=1}^k A_i = A$.
- If H is a processor, then $\delta(H) = \{H\}$. If H is a super-node, $\delta(H) = H$. If H a set of super-nodes H , $\delta(H)$ is the set of all processors in the super-nodes of H . If H consists of processors, super-nodes, and sets of super-nodes, $\delta(H) = \bigcup_{A \in H} \delta(A)$.
- For two sets of processors A, B , if there are $p \in A$ and $q \in B$ such that p, q do not like each other, then (A, B) is called a *faulty pair*. Otherwise, (A, B) is a *good pair*.
- A super-node is *good* if it does not contain faulty processor. A good super-node may contain both good and intermittent processors. A super-node is *intermittent* if it does not contain good processor. A *majority* super-node is a good super-node without any intermittent processor. A *majority* super-node is a faulty super-node without any intermittent processor.

We define some polynomial-bounded functions that will be used in Theorem 1 and its proof.

$$u(t) = 2^{\lceil \log(2t+1) \rceil}, \text{ where } \lceil a \rceil \text{ is the least integer larger than } a$$

$$w(t) = 100t + 201$$

$$\begin{aligned}
 x(t) &= u(t)w(t) \\
 y(t) &= x(t) + 2u(t) + t \cdot u(t) \\
 z(t) &= \frac{1}{2}y(t) + t \\
 p(t) &= t \cdot u(t) + 2u(t) \\
 q(t) &= \frac{1}{4}w(t)u(t) \\
 \epsilon(t) &= \frac{p(t)}{w(t)u(t)} < 0.01
 \end{aligned}$$

Lemma 1. Let A, B be sets of processors. Let $n = \max(|A|, |B|)$. Then

Proof: Let $A = \{a_1, \dots, a_m\}$ and $B = \{b_1, \dots, b_n\}$. Test as follows:

```

If  $m \leq n$  then
  for  $i := 0$  to  $n - 1$  do
     $j := 1$  to  $m$  do in parallel
       $a_j$  tests  $b_{(j+i) \bmod n}$ 
  else
  for  $i := 0$  to  $m - 1$  do
     $j := 1$  to  $n$  do in parallel
       $a_{(j+i) \bmod m}$  tests  $b_j$ 
    
```

Theorem 1. Let P be a set of n processors. If $|good(P)| > \frac{1}{2}n$ and $|intermittent(P)| \leq t$, then $faulty(P) \subseteq A \subseteq faulty(P) \cup intermittent(P)$ for some $A \subseteq P$ with $|A| = O(t^5)$.

$$faulty(P) \subseteq A \subseteq faulty(P) \cup intermittent(P).$$

Proof: If $n \leq 8z(t)(2t + 1)$, let each processor test every other processor; diagnose a processor as faulty iff a majority of the others say it is faulty. Clearly this algorithm diagnoses all but the intermittent processors correctly. The total number of rounds is $n \leq 8z(t)(2t + 1) = O(t^5)$.

Henceforth, we assume $n > 8z(t)(2t + 1)$. The remainder of our algorithm is based on the method of Beigel, Hurwood, and Kahale [1], who build “super-nodes,” each consisting of processors with identical fault status. However, in order to tolerate intermittent faults, we have made many modifications to their algorithm. Our algorithm is composed of 7 stages. We will write $r_i(t)$ to denote the number of rounds in Stage i .

In stage 1, we build up supernodes to size $u(t)$ such that every two processors in one supernode like each other. Each super node is either good or faulty. In other words, each super node does not contain both good and faulty processors, but it may contain both good and intermittent processors or both faulty and intermittent processors. The unpaired node is put into set J when forming the super node of size $2k$ from k .

In stage 2, we build up chains from the supernodes pairs that do not like each other so that every chain does not contain both good and faulty processors. A chain may contain both good and intermittent processors.

If the number of super nodes of size $u(t)$ is small, stage 3 is executed. Some large chains are selected. One of them will contain only good nodes. Build two sets $GOOD_i$ and $FAULTY_i$ from each chain C_i . The large chain C_i with only good super nodes can collect all of the good processors in its $GOOD_i$ and all of the faulty processors in its $FAULTY_i$. For a chain with only faulty super nodes, its $GOOD_i$ does not contain any good processor. We can finally judge which chain is good from the size of its $GOOD_i$.

If the number of super nodes of size $u(t)$ is large, we execute stages 4-8. Stage 4 applies the test graph to the super nodes of size $u(t)$. Select some giant sets G_1, \dots, G_m of super nodes of size $u(t)$. Each giant set only contain the same type of super nodes (good or faulty). At least one of them only contains good super nodes. Each G_i will build up its $GOOD_i$ and $FAULTY_i$ such that if G_i only contains good super nodes, its $GOOD_i$ will contain all good processors in P and its $FAULTY_i$ only contains all faulty processors in P after stages 5-8. If G_i only contains faulty super nodes, its $GOOD_i$ does not contain any good processor.

Stage 5 uses the giant sets to classify processors in all super nodes of size $u(t)$. Stage 6 uses the giant sets to classify all super nodes in the unpaired nodes in J . Stages 7 and 8 uses the giant sets to classify the chains.

Stage 1

```

let each processor constitute one super-node of size 1
k := 1
C := ∅
J := ∅
repeat
    if the number of super-nodes of size k is odd
    then put one of them into J
    arbitrarily pair up the others
    for each such pair ⟨A, B⟩ do
        let each p ∈ A and each q ∈ B test each other
        if all of those test results are good
        then let A ∪ B constitute a super-node of size 2k
        else C := C ∪ {⟨A, B⟩}
    k := 2k
until k = u(t)

```

End of Stage 1

Lemma 2

$$\begin{aligned}
 & \dots r_1(t) \leq 2(u(t) - 1) \\
 & \dots \\
 & \dots |\delta(J)| \leq 2u(t) - 1
 \end{aligned}$$

Proof: i. If A and B have size k , then all processors in A can test all processors in B in k rounds. The reverse tests can be performed in an additional k rounds. Thus the total number of rounds for Stage 1 is $2 \cdot 2^0 + 2 \cdot 2^1 + \dots + 2 \cdot (u(t)/2) = 2(u(t) - 1)$.

ii. Suppose A is a smallest super-node that contains a good processor p and a faulty processor q . A must be formed from the union of two super-nodes in a pair $\langle A', B' \rangle$. Since A is the smallest super-node, p and q cannot be at the same super-node in $\{A', B'\}$. Hence it is impossible for p and q like each other. This is a contradiction.

iii. For each k in Stage 1, J contains at most one super-node of size k . Hence, $|\delta(J)| \leq 2^0 + 2 + 2^2 + \dots + u(t) = 2u(t) - 1$. ■

After Stage 1, let S be the set of all super-nodes of size $u(t)$; J be the set of all unpaired super-nodes; and C be the set of all faulty pairs. Arrange all pairs in C sequentially: $\langle A_{1,0}, A_{1,1} \rangle, \langle A_{2,0}, A_{2,1} \rangle, \dots, \langle A_{k,0}, A_{k,1} \rangle$.

Definition 2

- A \dots is a series of super-nodes $A_{i,a_i}, A_{i+1,a_{i+1}}, \dots, A_{j,a_j}$ in the pairs of C , where $a_m \in \{0, 1\}$ for $i \leq m \leq j$. Such a chain does not contain both good and faulty processors.
- For two processors p, q in C , let $\text{dist}(p, q)$ denote the number of pairs between p and q . In other words, if $p \in A_{i,a_i}$ and $q \in A_{j,a_j}$, $\text{dist}(p, q) = |j - i| - 1$.
- A pair in C is \dots if at least one of its two super-nodes contains intermittent processors.
- A chain K is \dots (resp., \dots) if it contains only good (resp., faulty) processors.
- A pair in C is \dots if both of its super-nodes contain only faulty processors.
- A pair in C is \dots if one of its super-nodes contains only faulty processors and the other one contains only good processors.
- For a chain K in C , if K begins with super-node $A_{i,a}$ and ends with $A_{j,b}$, then the pair $\langle A_{i-1,0}, A_{i-1,1} \rangle$ and $\langle A_{j+1,0}, A_{j+1,1} \rangle$ are called the \dots and \dots of K respectively.
- For a chain H in C and a super-node A in a pair of C , we say $H \dots A$ if for every processor p in the super-nodes of H , and every q in A such that $\text{dist}(p, q) \leq t + 1$, p and q like each other.

We partition the super-nodes in C into chains via a simple greedy algorithm:

Stage 2

for all p, q in C with $\text{dist}(p, q) \leq t + 1$ do

let p and q test each other

$C_0 := \{A_{1,0}\}$

$C_1 := \{A_{1,1}\}$

$i := 1$

Chains $:= \emptyset$

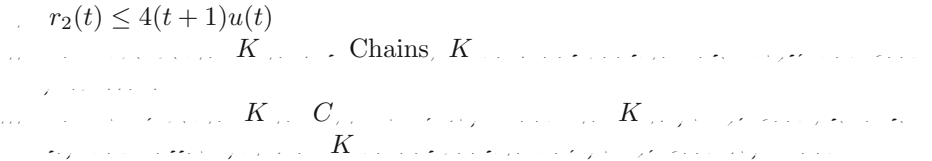

```

repeat
  i := i + 1
  if C0 likes Ai,j for some j ∈ {0, 1} then
    choose such a j
    C0 := C0 ∪ {Ai,j}
    if C1 likes Ai,1-j then
      C1 := C1 ∪ {Ai,1-j}
    else
      Chains := Chains ∪ {C1}
      C1 := {Ai,1-j}
  else
    Chains := Chains ∪ {C0}
    if C1 likes Ai,j for some j ∈ {0, 1} then
      choose such a j
      C1 := C1 ∪ Ai,j
      C0 := {Ai,1-j}
    else
      Chains := Chains ∪ {C1}
      C0 := {Ai,0}
      C1 := {Ai,1}
until i = k

```

End of Stage 2

Lemma 3



Proof: i. The tests can be performed in $t+1$ substages. In the i th substage, every processor in $A_{j,0} \cup A_{j,1}$ test each other with every processor in $A_{j+i,0} \cup A_{j+i,1}$ for $1 \leq j \leq k-i$. By Lemma 1, $r_2(t) \leq 4(t+1)u(t)$.

ii. Assume K contains a good processor and a faulty processor. Since there are at most t intermittent processors, there are a faulty processor p and a good processor q in K with $\text{dist}(p, q) \leq t+1$. Hence p, q do not like each other and will not be in the same chain of C .

iii. Suppose the top pair of K contains a purely good super-node. Then K has to contain a purely good super-node from its top pair in Stage 2. This is a contradiction. Similarly, the bottom pair does not contain a purely good super-node. █

Let C_{chains} be the set of all chains built at stage 2. The Lemma 3 shows every chain in C_{chains} is either good or faulty. We consider two cases which depend on the size of S , where S is the set of all super-node of size $u(t)$ as defined before.

Case 1: $|S| \leq w(t)$ The number of processors in S is no more than $u(t)w(t) = x(t)$. The number of processors in the super-nodes in J is no more than $2u(t)$. The number of processors in special pairs is bounded by $tu(t)$. The number of processors in J , S and special super-nodes in $C \leq x(t) + 2u(t) + tu(t) = y(t)$. Since more than half processors are good, the number of processors in faulty-faulty pairs in C is $\leq y(t)$. Good chains are separated by faulty-faulty pairs and special pairs. The number of faulty-faulty pairs is at most $\frac{1}{2}y(t)$. Hence the number of faulty-faulty pairs and special pairs is at most $\frac{1}{2}y(t) + t = z(t)$. Since the number of good processors is more than $\frac{1}{2}n$, there is a chain K in C_{chains} that contains at least $\frac{\frac{1}{2}n - y(t)}{z(t) + 1} \geq \frac{n}{4z(t)}$ good processors. The total number of chains with more than $\frac{n}{4z(t)}$ processors is no more than $4z(t)$. Let K_1, \dots, K_m be all of the chains in C with at least $\frac{n}{4z(t)}$ processors, where $m \leq 4z(t)$.

Stage 3

for $i := 1$ to m do

 GOOD $_i := \delta(K_i)$

 partition $\delta(K_i)$ into A_1, \dots, A_{r+1} such that $|A_i| = 2t + 1$ for $1 \leq i \leq r$

and $|A_{r+1}| \leq 2t + 1$

 partition $P - \delta(K_i)$ into P_1, \dots, P_r such that $||P_j| - |P_k|| \leq 1$ for each

j, k

 for $j := 1$ to r do

 for each $p \in P_j$ do

 for each q in A_j do

 let p and q test each other

 if p and most q 's in A_j like each other then

 put p into GOOD $_i$

 if $|\text{GOOD}_i| > \frac{1}{2}n$ then output $A = P - \text{GOOD}_i$.

End of Stage 3

Lemma 4. $r_3(t) \leq 64z(t)^2(2t + 1)$

... K_i ... $\delta(K_i)$...
 ...
 ... K_i ... $\delta(K_i)$...

Proof: (i) Since each K_i has at least $\frac{n}{4z(t)}$ elements and $n > 8z(t)(2t + 1)$,

$$r \geq \frac{n}{4z(t)(2t + 1)} - 1 \geq \frac{n}{8z(t)(2t + 1)}.$$

Therefore, $|P_i| \leq 8z(t)(2t + 1)$. For each $i \leq m$, we use at most $16z(t)(2t + 1)$ rounds by Lemma 1. Hence Stage 3 takes at most

$$m \cdot 16z(t)(2t + 1) \leq 64z(t)^2(2t + 1)$$

rounds in all.

(ii) If K_i is a good chain, K_i does not contain any faulty processor. For a processor $p \in P - \delta(K_i)$, p can not be faulty if the most results are good

among the $2t + 1$ mutual tests with every processors in some $A_j \subseteq \delta(K_i)$ and $|A_j| = 2t + 1$.

(iii) Similar to the proof of (ii). ■

Case 2: $|S| > w(t)$ Since the number of processors in special pairs and super-nodes of J is $\leq tu(t) + 2u(t) = p(t)$, the number of good processors in the pure good super-nodes of S is at least

$$\frac{1}{2}|S|u(t) - p(t) \geq \left(\frac{1}{2} - \epsilon(t)\right)|S|u(t) > 0.49|S|u(t) > q(t). \tag{1}$$

Thus, S contains at least $0.49|S|$ pure good super-nodes.

Lemma 5. Let $0 < \gamma, \lambda < 1$. Let N be a super-node of H_d with m processors. Let N' be a super-node of H_d with λm processors. Let N'' be a super-node of H_d with $\gamma \lambda m$ processors.

$$2^{1+\lambda} \left(2 \frac{(m - \alpha m)!(m - \beta m)!}{m!(m - \alpha m - \beta m)!} \right)^d,$$

where $\alpha = \frac{(1+\gamma)}{2}m$ and $\beta = \frac{(1-\gamma)}{2}m$.

Stage 4

Identify the elements of S with nodes in H_{16} .

Construct another graph G with vertex set S as follows:

for every two super-nodes A, B of S with an edge from A to B in H_{16}

 let A and B perform a $u(t)$ -mutual test

 if the test result is $(u(t), u(t)/2)$ good

 then place an edge an edge from A to B in G

Partition S according to the strongly connected components $G'_1, \dots, G'_{m'}$ in G .

End of Stage 4

Lemma 6

Let G' be a strongly connected component of G with $r_4(t) = 64$ super-nodes.

Proof: i. Suppose G' has a good super-node N_0 . Let N be a super-node of G' . There is a path $N_0, N_1, \dots, N_k = N$ in G' since G' is a strongly connected component. Each super node in S can not contain both good and faulty processors. If N_1 is a faulty super-node, there are at least $\frac{1}{2}u(t) \geq t + 1$ intermittent processors in $N_0 \cup N_1$ since N_0 is good and the $u(t)$ -mutual test between them is $(u(t), \frac{1}{2}u(t))$ -good. Therefore, N_1 has to be good. Similarly, we can show that $N_2, \dots, N_k = N$ are all good.

ii. A single $u(t)$ -mutual test can be performed in two rounds. For a Hamiltonian path with nodes in S , it takes 4 rounds to take $u(t)$ -mutual tests for the super nodes in S . For 16 Hamiltonian path, it needs $4 \times 16 = 64$ rounds. Therefore, $r_4(t) = 64$. █

Let G_1, \dots, G_m be all the giant nodes among $G'_1, \dots, G'_{m'}$ in S . Then each $|G_i| > \frac{1}{6}|S|$ and $m \leq 5$ (For the details of the probability and size calculation based on Lemma 5, see [2],[6]). Since there are more than $0.49|S|u(t)$ good processors in S , by Lemma 5, one of the giant nodes, say G_{i_0} , only contains good super-nodes.

Stage 5

for $i := 1$ to m do

GOOD $_i :=$ FAULTY $_i := \emptyset$

Let B_1, \dots, B_r be super-nodes in G_i .

Partition $S - G_i$ into Q_1, \dots, Q_r such that $||Q_k| - |Q_j|| \leq 1$ for $1 \leq$

$k, j \leq r$.

For each super-node $D' \in Q_j$, let D', B_j take $u(t)$ -mutual test.

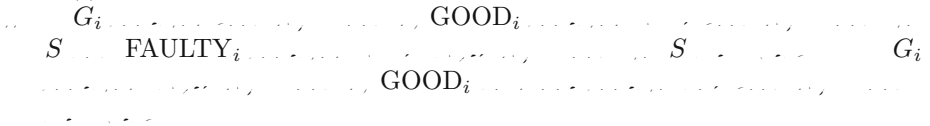
If most results are good, put D' into GOOD $_i$

else put D' into FAULTY $_i$.

End of Stage 5

Lemma 7

$$r_5(t) \leq 50$$



Proof: i. Since $|S - G_i| \leq \frac{5}{6}|S|$ and $|G_i| \geq \frac{1}{6}|S|$, we have $|Q_j| \leq 5$ for $1 \leq j \leq r$. Therefore, each B_j takes at most $2 \cdot 5$ rounds to process Q_j . Thus the total number of rounds is at most 50.

ii. Every super-node N in $S - G_i$ takes $u(t)$ -mutual test with a super-node in G_i . If most results are good, N is put into GOOD $_i$. Otherwise, N is put into FAULTY $_i$. For two super-nodes A_1, A_2 of size $u(t)$, if they are both good, most results will be good in the $u(t)$ -mutual test between. This is because there are no more than t intermittent processors in $A_1 \cup A_2$ and $u(t) \geq 2t + 1$. It is easy to see that if one of A_1, A_2 is faulty and the other is good, most results are faulty in the $u(t)$ -mutual test between them. █

There is a GOOD $_j$ containing all good super-nodes in S which is more than $0.49|S|$. We use the good processors in S to find out all good processors in J and C . The method is similar to the method in [1]. As the number of good processors in S is big enough, it can be done in small number of parallel rounds.

Let b be the number of faulty-faulty pairs in C , the set of paired super-nodes from Stage 1. We can find at most $2(b + t) + 1$ chains to cover all C

(every pair in C has one of its two super-nodes in one of the those chains). Let $D = \{D_1, \dots, D_c\} \subseteq C_{\text{chains}}$ be the set of chains covering C , where $c \leq 2(b+t)+1$. D can be determined via greedy algorithm, i.w., we choose a chain from C covering largest number of uncovered pairs until C is covered.

Let x be the number of good processors in S . Since most processors in P are good and there are at most t super-nodes in S containing intermittent processors, we have $x + |\delta(J)| + tu(t) \geq 2b$. Therefore, $x \geq 2b - 2u(t) - tu(t)$.

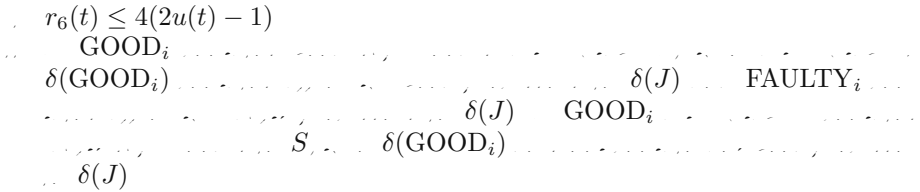
Without lose generality, let $\text{GOOD}_1, \dots, \text{GOOD}_d$ are of size $\geq 0.49|S|$ ($d = 1$ or 2). One of the GOOD_i 's, say GOOD_{i_0} , does not contain any super-node with faulty processors and has $\geq q(t)$ good processors (by inequality (1)).

Stage 6

- for $i = 1$ to d do
 - let B be a super-node in G_i
 - let every processor p in $\delta(J)$ take mutual test with every processor in B .
 - if p likes each other with most processors in B
 - then put p into GOOD_i
 - else put p into FAULTY_i .

End of Stage 6

Lemma 8



Proof: i. Since $|\delta(J)| \leq 2u(t) - 1$ (by Lemma 2) and $|B| = u(t)$, Stage 6 costs no more than $2(2u(t) - 1)$ for any fixed i (by Lemma 1). Therefore, $r_6(t) \leq 4(2u(t) - 1)$ since $d \leq 2$.

ii. If B is a good super-node of size $u(t)$, most results are good when all processors in B test a good processor in $\delta(J)$, and most results will be faulty when all processors in B test a faulty processor. This is because $u(t) \geq 2t + 1$ and there are no more than t intermittent processors in B . ■

Stage 7

- (1) for $i := 1$ to d do
- (2) let B_1, \dots, B_{m_i} be the super-nodes in GOOD_i .
- (3) let F_1 be the set of all chains $D_j \in D$ with $|D_j| < u(t)$.
- (4) let F_2 be the set of all chains $D_j \in D$ with $|D_j| \geq u(t)$.
- (5) partition F_1 into $F_{1,1}, \dots, F_{1,m_i}$ such that $||F_{1,l}| - |F_{1,k}|| \leq 1$ for $1 \leq l, k \leq m_i$.
- (6) partition F_2 into $F_{2,1}, \dots, F_{2,m_i}$ such that $||F_{2,l}| - |F_{2,k}|| \leq 1$ for $1 \leq l, k \leq m_i$.
- (7) if $|F_{a,b}| \leq 4u(t)$ for $1 \leq a \leq 2$ and $1 \leq b \leq m_i$ then
- (8) for $j := 1$ to m_i do in parallel

- (9) every processor in B_j take mutual test with every processor $q \in \delta(F_{1,j})$.
 if q likes each other with most processors in B_j
 then put q into GOOD_i .
 else put q into FAULTY_i .
- (10) for $j := 1$ to m_i do in parallel
- (11) let B_j take $u(t)$ -mutual test with every chain $D_k \in F_{2,j}$.
 if most results are good
 then put D_k into GOOD_i
 else put D_k into FAULTY_i .

End of Stage 7

Lemma 9

$$r_7(t) = O(t^2)$$

$$\begin{array}{c} \text{GOOD}_i \dots \dots \dots S \dots \dots \dots \\ \text{good}(\delta(D)) \subseteq \delta(\text{GOOD}_i) \quad \text{faulty}(\delta(D)) \subseteq \delta(\text{FAULTY}_i) \quad \text{GOOD}_i \dots \dots \dots \\ \dots \dots \dots \delta(D) \dots \dots \dots \delta(\text{GOOD}_i) \dots \dots \dots \end{array}$$

Proof: i. Assume $|F_{a,b}| \leq 4u(t)$. We have $|\delta(F_{1,j})| \leq |F_{1,j}|u(t) \leq 4u(t)^2$. The loop ((8)–(9)) costs $12u(t)^2$ rounds (by Lemma 1). Since each D_k takes 2 rounds for the mutual test with B_j , the loop ((10)–(11)) costs $\leq 12u(t)$ rounds. Since $d \leq 2$, we have $r_7(t) \leq 2(12u^2(t) + 12u(t))$.

ii. If $b > q(t) - t$, then $x \geq 2b - 2u(t) - tu(t) > b + t$. If $b < q(t) - t$, then $x > q(t) > b + t$ (by inequality (1)). Therefore, $x > b + t$ holds in every case. Suppose GOOD_i contains all good super-nodes in S . $m_i \geq \frac{x}{u(t)} > \frac{b+t}{u(t)}$ super-nodes from S . Since $c \leq 2(b + t) + 1 < 3(b + t)$, we have

$$|F_{1,j}|, |F_{2,j}| \leq \frac{c}{m_i} + 1 \leq \frac{3(b+t)}{m_i} + 1 \leq 3u(t) + 1 < 4u(t) \text{ for } j = 1, \dots, m_i.$$

Therefore, the condition at the if statement (7) is satisfied. It is easy to see that every good processor in $\delta(D)$ is in $\delta(\text{GOOD}_i)$ and every faulty processor in $\delta(D)$ is in $\delta(\text{FAULTY}_i)$. █

Stage 8

- (1) for $i := 1$ to d do
- (2) let B_1, \dots, B_{m_i} be the super-nodes of size $u(t)$ in GOOD_i .
- (3) let $F_1 = \{D_j : D_j \in (\text{GOOD}_i \cap D) \text{ and } |D_j| \geq u(t)\}$ and $F'_1 = \{K : \text{partner}(K) \in D_j \text{ for some } D_j \text{ in } F_1\}$.
- (4) for each $D_j \in F_1$ do in parallel
- (5) let $D_{j,1}, \dots, D_{j,i_j+1}$ be a $u(t)$ -partition of $\delta(D_j)$.
- (6) let $D'_j = \{K : \text{partner}(K) \in D_j\}$
- (7) partition $\delta(D'_j)$ into $E_{1,1}, \dots, E_{1,i_j}$ such that $\|E_{1,l}\| - \|E_{1,k}\| \leq 1$.
- (8) for $l = 1$ to i_j do in parallel

- (9) let every processor $p \in E_{1,l}$ take mutual test with every processor in $D_{j,l}$.
 if p likes each other with most processors in $D_{j,l}$
 then put p into GOOD_i
 else put p into FAULTY_i .
- (10) let $F_2 = \{D_j : D_j \in D \text{ and } \delta(D_j) \subseteq \delta(\text{GOOD}_i) \text{ and } |D_j| < u(t)\}$,
 and
 let $F'_2 = \{K : \text{partner}(K) \in D_j \text{ for some } D_j \in F_2\}$
- (11) partition $\delta(F'_2)$ into $E_{2,1}, \dots, E_{2,m_i}$ such that $||E_{2,l}| - |E_{2,k}|| \leq 1$
 for $1 \leq l, k \leq m_i$.
- (12) let N_0 be the set of all super-nodes in the chains in $C - D$
- (13) let $N_1 = (N_0 - F_1) - F_2$.
- (14) let F_3 be the set of all chains M such that M contains some super-nodes in N_1 and
 $|M| < u(t)$.
- (15) let F_4 be the set of all chains M such that M contains some super-nodes in N and
 $|M| \geq u(t)$.
- (16) partition $\delta(F_3)$ into $E_{3,1}, \dots, E_{3,m_i}$ such that $||E_{3,l}| - |E_{3,k}|| \leq 1$
 for $1 \leq l, k \leq m_i$.
- (17) partition F_4 into $E_{4,1}, \dots, E_{4,m_i}$ such that $||E_{4,l}| - |E_{4,k}|| \leq 1$
 for $1 \leq l, k \leq m_i$.
- (18) if $|E_{a,b}| \leq 4u(t)$ for $2 \leq a \leq 4$ and $1 \leq b \leq m_i$ then
- (19) for $j := 1$ to m_i do in parallel
- (20) let every processor $p \in E_{2,j}$ take mutual test with every processor in B_j .
 if most results are good
 then put p into GOOD_i
 else put it into FAULTY_i .
- (21) for $j := 1$ to m_i do in parallel
- (22) let every processor $p \in E_{3,j}$ take mutual test with every processor in B_j .
 if most results are good
 then put p into GOOD_i
 else put it into FAULTY_i .
- (23) for $j := 1$ to m_i do in parallel
- (24) let each chain $D_k \in E_{4,j}$ take $u(t)$ -mutual test with B_j .
 if most results are good
 then put D_k into GOOD_i
 else put D_k into FAULTY_i .
- (25) if $|\delta(\text{GOOD}_i)| > \frac{n}{2}$, output $A = \delta(\text{FAULTY}_i)$.

End of Stage 8

Lemma 10

$$r_8(t) = O(t^2)$$

GOOD_{*i*}

$$H = \{K : \text{partner}(K) \dots D_j \in D \dots$$

$$(D_j \in \text{FAULTY}_i \dots \text{partner}(K) \in \text{FAULTY}_i \dots K \dots \text{FAULTY}_i)\}$$

$$\begin{aligned} & \dots 2(b+t) + 1 \dots C_{\text{chains}} - D \\ \text{GOOD}_i & \dots S \dots \text{good}(\delta(C_{\text{chains}} - D)) \subseteq \\ \delta(\text{GOOD}_i) & \dots \text{faulty}(\delta(C_{\text{chains}} - D)) \subseteq \delta(\text{FAULTY}_i) \\ \text{FAULTY}_i & \dots S \dots \text{good}(\delta(\text{FAULTY}_i)) \\ & = \emptyset \end{aligned}$$

Proof: i. Let us fix an integer *i* at the first loop of Stage 8.

Phase 1 ((8)-(9)): Since $|\delta(D_j)| = |\delta(D'_j)| \geq u(t)$, $|E_{1,k}| \leq 2u(t)$. This phase costs $\leq 4u(t)$ rounds (by Lemma 1).

Phase 2 ((19)-(20)): Since $|E_{2,k}| \leq 4u(t)$, this phase costs $\leq 8u(t)^2$ rounds (by Lemma 1).

Phase 3 ((21)-(22)): Since $|E_{3,k}| \leq 4u(t)$, this phase costs $\leq 8u(t)^2$ rounds (by Lemma 1).

Phase 4 ((23)-(24)): Since $|E_{4,k}| \leq 4u(t)$, this phase costs $\leq 8u(t)$ rounds.

ii. Suppose GOOD_{*i*} contains all super-nodes in *S*. For every super-node *K* ∈ *H*, partner(*K*) is not a purely good super-nodes. Each super-node of *H* is either in a faulty-faulty pair, special pair, or a good chain in *C*_{chains} − *D*. The set *C* has $\leq b+t$ mixed super-nodes and pure faulty super-nodes which are in faulty–faulty pairs. Good chains are separated by mixed pairs and faulty–faulty pairs. Thus, the super-nodes of *H* are in no more than $2(b+t) + 1$ chains in *C*_{chains} − *D*.

iii. As the proof of Lemma 9, we have $m_i \geq \frac{b+t}{u(t)}$ and $|E_{2,k}|, |E_{3,k}|, |E_{4,k}| \leq 4u(t)^2$. Therefore the condition at if (18) can be always satisfied. Everything else is similar. █

Combining those lemmas, we have the total number of rounds is $O(t^5)$ and $\text{faulty}(P) \subseteq A \subseteq \text{faulty}(P) \cup \text{intermittent}(P)$. █

4 Conclusion

In this paper we give a $O(t^5)$ parallel rounds algorithm to tolerate at most *t* intermittent processors. We feel there is still some space to decrease the upper bound. An interest open problem is deriving non-linear lower bound for tolerating *t* intermittent processors.

Acknowledgement

We would like to thank the anonymous referees for their helpful comments to the earlier version of this paper.

References

1. R. Beigel, W. Hurwood, and N. Kahale. Fault diagnosis in a flash. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 571–580, 1995.
2. R. Beigel, R. Kosaraju, and G. Sullivan. Locating faults in a constant number of parallel testing rounds. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 189–198, 1989.
3. R. P. Bianchini, Jr. and R. Buskens. An adaptive distributed system-level diagnosis algorithm and its implementation. In *Proceedings 21st Int. Symp. Fault Tolerant Computing*, pages 222–229, 1991.
4. R. P. Bianchini, Jr., K. Goodwin, and D. S. Nydick. Practical application and implementation of distributed system-level diagnosis theory. In *Proceedings 20th Int. Symp. Fault Tolerant Computing*, pages 332–339, 1990.
5. W. Hurwood. Ongoing diagnosis. In *Proceedings 15th IEEE Symp. Reliable Distributed Systems*, 1996.
6. W. Hurwood. *System Level Fault Diagnosis under Static, Dynamic, and Distributed Models*. PhD thesis, Yale University, Dept. of Computer Science, New Haven, CT, 1996.
7. U. Manber. System diagnosis with repair. *IEEE Trans. Comput.*, C-29:934–937, 1980.
8. K. Nakajima. A new approach to system diagnosis. In *Proceedings 19th Ann. Allerton Conf. Commun. Contr. and Comput.*, pages 697–706, 1981.
9. F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Trans. Electron. Comput.*, EC-16:848–854, 1967.

Three-Round Adaptive Diagnosis in Binary n -Cubes

Satoshi Fujita^{1,*} and Toru Araki²

¹ Department of Information Engineering,
Graduate School of Engineering, Hiroshima University

² Department of Computer and Information Sciences,
Faculty of Engineering, Iwate University

Abstract. In this paper, we consider the problem of adaptive fault-diagnosis in binary n -cubes, and propose a scheme that completes a diagnosis in at most three test rounds, provided that $n \geq 3$ and the number of faulty vertices is at most n . The proposed scheme is optimal in the sense that: 1) three rounds are necessary for the adaptive diagnosis, 2) there exists a set of $n + 1$ faulty vertices that can not be located by any diagnosis scheme, and 3) to identify n faulty vertices, the system must contain at least $2n + 1$ vertices. Note that $n = 3$ is the smallest integer satisfying $2^n \geq 2n + 1$.

1 Introduction

The system-level fault-diagnosis is one of the main issues to realize highly dependable parallel processing systems. The concept of *adaptive diagnosis* has first been proposed by Preparata, Metze, and Chien [12]. They consider a multiprocessor system that could be modeled as a collection of processing elements called **units**, and assume that each of the units has the capability of testing the other units to report the test results to a central observer. By analyzing the complete collection of the test results, the observer identifies which unit is *faulty* and which is *fault-free*. A key point of the Preparata *et al.*'s diagnosis model, that is commonly referred to as the *Preparata-Chien model* in the literature, is that the test result reported by a faulty unit is unreliable, whereas the result reported by a fault-free unit is always correct.

In the original PMC model, the test phase is conducted exactly once; thus the main objective of the problem is to identify a minimum number of faulty units that is consistent with the given collection of the test results (note that the minimality is necessary to exclude trivial answers such that "all units are faulty"). A system is said to be t -diagnosable if there exists a collection of tests such that all faulty units are identified, provided that the number of faulty units

* This research was partially supported by the Japan Society for the Promotion of Science Grant-in-Aid for Scientific Research (C), 13680417, and Priority Areas (B)(2) 16092219.

does not exceed t . It is well known that a t -diagnosable system must contain at least $2t + 1$ units, and each unit of the system must be tested by at least t other units; thus it needs at least Nt tests altogether where N is the number of units in the system [12]. A characterization of t -diagnosable systems is given by Hakimi and Amin [5].

In [9], Nakajima proposed an adaptive diagnosis model to improve the efficiency of the diagnosis process. An adaptive diagnosis process proceeds in several test rounds, and the objective of the problem is to reduce the number of rounds as well as the total number of tests. It has been shown that by applying the adaptive model to a completely connected system, the number of tests that is necessary and sufficient to identify at most t faults reduces from Nt to $N + t - 1$ [3] and the number of test rounds that is necessary to identify all faults could be bounded by a constant even when simultaneous participation of a unit to several tests is not allowed [1, 6]. The power of adaptive diagnosis has also been investigated for several classes of sparse network topologies such as trees, cycles, and tori [7], butterfly networks [11], and cube-connected cycles [10]. Among them, Feng et al. [4] considered the adaptive fault-diagnosis in binary n -cubes with at most n faults, and proposed an $(n + 4)$ -round algorithm with at most $2^n(\lceil \log n \rceil + 2)$ tests (note that since an n -cube can be factorized into n perfect matchings, a trivial upper bound is $2n$ rounds with $n2^{n+1}$ tests). This result was improved to 11 rounds with at most $2^n + 3n/2$ tests by Kranakis and Pelc [7], and to four rounds by Björklund [2] (in addition to this improvement, Björklund proved that for the adaptive diagnosis of n -cubes, $2^n + n - 1$ tests are necessary and sufficient). Note that a lower bound on the number of test rounds is three [2]. Recently, Nomura et al. showed that three test rounds are necessary and sufficient if the number of faults is at most $n - \lceil \log(n - \lceil \log \log n \rceil + 4) \rceil + 2$ [10].

In this paper, we prove that for any $n \geq 3$, an adaptive diagnosis in binary n -cube takes at most three test rounds, provided that the number of faults does not exceed n . The proposed scheme is optimal in the sense that: 1) three rounds are necessary, 2) there exists a fault set with $n + 1$ units that can not be identified by any scheme, and 3) to identify n faulty units, the system must contain at least $2n + 1$ units; i.e., $n = 3$ is the smallest integer satisfying $2^n \geq 2n + 1$.

The remainder of this paper is organized as follows. In Section 2, a formal definition of the adaptive diagnosis model and binary n -cubes will be provided. The proposed three-round scheme is given in Section 3. Section 4 concludes the paper with future problems.

2 Preliminaries

2.1 Diagnosis Model

Let $G = (V, E)$ be an undirected graph that models the topology of an underlying parallel processing system, where vertices in V represent processors and edges in E represent communication links between them. A vertex can test the status of its adjacent vertices via the communication link connecting them. A **test** of v by u is denoted by an ordered pair (u, v) , and an outcome of the test is denoted

by $r(u, v)$; i.e., $r(u, v) = 0$ if u judges v is fault-free and $r(u, v) = 1$ if u judges v is faulty. It should be noted that the result of a test conducted by a faulty unit is unreliable, whereas the result of a test conducted by a fault-free unit is always correct.

In this paper, we assume that a diagnosis process proceeds in several rounds, and in each round, each vertex can participate at most one test; i.e., if two tests (u, v) and (x, y) are conducted in the same round, four vertices u, v, x and y must be distinct. A **test assignment** is a set of tests that is conducted in a round. By definition, a test assignment for $G = (V, E)$ is a matching in directed graph $(V, \{(u, v) \mid \{u, v\} \in E\})$. A diagnosis process is a sequence of test assignments. In this paper, we assume that there is an **observer** who can collect all test results conducted by the vertices. A collection of test results with respect to a given test assignment is referred to as a **syndrome**. The main role of the observer is, given a sequence of syndromes, to determine the test assignment to be conducted in the next round, as well as the identification of faulty vertices from the syndromes. In the following, for brevity, a test with result 0 (resp. 1) is referred to as a 0-arrow (resp. 1-arrow).

2.2 Binary n -Cube

A binary n -cube $Q_n = (V(Q_n), E(Q_n))$ is a bipartite graph constructed as follows: 1) each vertex in $V(Q_n)$ corresponds to a binary string of length n , and 2) two vertices in $V(Q_n)$ are connected by an edge iff the corresponding binary strings differ in exactly one bit position. It is widely known that Q_n is Hamiltonian. A typical way to construct a Hamiltonian cycle in Q_i is described as follows:

```

function Hamilton( $i$ : dimension)
begin
if  $i = 1$  then return  $(0, 1)$  as  $H_i$ 
else begin
Let  $H_{i-1} = (v_1, v_2, \dots, v_{2^{i-1}})$  be the output of Hamilton( $i - 1$ );
return  $(0v_1, 0v_2, \dots, 0v_{2^{i-1}}, 1v_{2^{i-1}}, 1v_{2^{i-1}-1}, \dots, 1v_1)$  as  $H_i$ 
end
end.

```

By using the procedure, $H_2, H_3,$ and H_4 are constructed as follows:

$$\begin{aligned}
 H_2 &= (00, 01, 11, 10) \\
 H_3 &= (000, 001, 011, 010, 110, 111, 101, 100) \\
 H_4 &= (0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100 \\
 &\quad 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000)
 \end{aligned}$$

By construction, H_i is a Hamilton path in Q_n and the first and the last vertices in H_i are adjacent in Q_n . In addition, H_i is a concatenation of two Hamilton

paths in two copies of Q_{n-1} ; i.e., the first one is a subcube of Q_n to have prefix 0 and the latter is a subcube of Q_n to have prefix 1.

In the next section, we propose an adaptive diagnosis scheme with three rounds. In the proposed scheme, we repeatedly use the following simple lemma:

Lemma 1. *Let $U \subseteq V(Q_n)$ be a subset of vertices in an n -cube Q_n . Let $W \subseteq V(Q_n) - U$ be a subset of vertices in Q_n . Then there is a matching of size $|U|$ between U and W in Q_n .*

Since Q_n is bipartite, $V(Q_n)$ can be partitioned into two subsets, say V_1 and V_2 , such that every edge in $E(Q_n)$ connects a vertex in V_1 with a vertex in V_2 . Let $U_i = U \cap V_i$ for $i = 1, 2$. Assume $U_1 \neq \emptyset$, without loss of generality. Since the degree of Q_n is n , each vertex in U_1 is connected with at least $n - |U_2|$ vertices in $V(Q_n) - U$. Since $n - |U_2| = |U_1|$, by Hall's theorem, there is a subset $W_1 (\subseteq V(Q_n) \cap V_2)$ of cardinality at least $|U_1|$ such that there is a matching of size $|U_1|$ between U_1 and W_1 in Q_n . Hence the lemma follows.

3 Scheme

3.1 First Partitioning

Suppose $n \geq 3$. Let $m = \lceil \log_2(n + 1) \rceil$. Note that $2^m \geq n + 1$ and $m < n$, for any $n \geq 3$. We first partition the given n -cube Q_n into 2^{n-m} small m -cubes, $P_1, P_2, \dots, P_{2^{n-m}}$, where vertices in each m -cube is identified by the prefix of length $n - m$. We then construct a Hamiltonian cycle in each m -cube by function **Hamilton** given in the last section. In the first two rounds, we conduct tests along those 2^{n-m} Hamiltonian cycles independently in parallel. Since $|V(Q_m)| \geq n + 1$, any m -cube with no 1-arrows must be fault-free. Hence, at most n subcubes can contain 1-arrows, since we are assuming that there are at most n faulty vertices. Let x denote the number of m -cubes containing 1-arrows, and let $\mathcal{Q} = \{P_1, P_2, \dots, P_x\}$ be the set of those m -cubes.

3.2 Overview of the Scheme

If $x \leq n - m$, we can complete the diagnosis by spending one more round, in the following manner: Consider an $(n - m)$ -cube \hat{Q}_{n-m} whose vertices correspond to the 2^{n-m} m -cubes constructed above. Let U be a subset of $V(\hat{Q}_{n-m})$ consisting of x vertices corresponding to m -cubes in \mathcal{Q} . Since $x \leq n - m$, by Lemma 1, there is a set $W \subseteq V(\hat{Q}_{n-m}) - U$ such that there is a matching of size $|U|$ between U and W in $V(\hat{Q}_{n-m})$. Since each m -cube corresponding to a vertex in W have been known to be fault-free in the first two rounds, by using (parallel) test arcs connecting those m -cubes, we can complete the fault-diagnosis by spending one more round. Hence in the following, we will assume that it holds $n - m + 1 \leq x \leq n$, without loss of generality.

Each m -cube in \mathcal{Q} contains at least one and at most m faulty vertices (in fact, if it contains more than m faulty vertices, the total number of faulty vertices

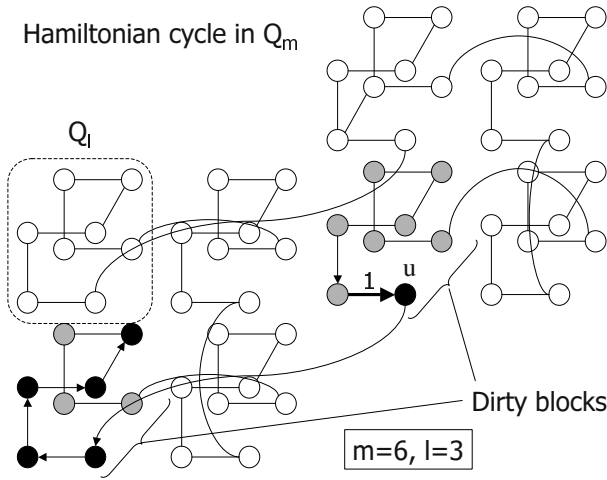


Fig. 1. Dirty blocks

becomes at least $m+1+(x-1) \geq n+1$, a contradiction). This implies that for each 1-arrow, we may check the fault-freeness of the consecutive m vertices starting from the head of the 1-arrow. This idea could be generalized by introducing m' to the overall system to indicate that for each 1-arrow, we may check the fault-freeness of m' consecutive vertices from the head of the arrow. Parameter m' is initialized to m , and during the execution of the algorithm, it will be decreased monotonically or it will be guaranteed (in some way) that the check for the current m' completes by spending one more round.

3.3 Reduction

In this subsection, we assume $m \geq 5$. The discussion about smaller m 's will be given in the next subsection. Given parameter m' , let $\ell = \lceil \log_2 m' \rceil$. We partition each m -cube in \mathcal{Q} into $2^{m-\ell}$ smaller ℓ -cubes (note that $m-\ell \geq m - \lceil \log_2 m \rceil \geq 2$ for any $m \geq 4$). Consider a Hamiltonian cycle constructed in the first two rounds. A (consecutive) part of the Hamiltonian cycle contained in an ℓ -cube is called a **block**, and a block is said to be “dirty” if it contains vertices to be tested (note that each Hamiltonian cycle is divided into $2^{m-\ell}$ blocks). Figure 1 illustrates dirty blocks in a Hamiltonian cycle of 6-cube for $m' = 6$ and $\ell = 3$. In this example, we have to check six ($= m'$) consecutive vertices painted black that is starting from a vertex pointed by a 1-arrow, i.e., vertex u . Note that any non-dirty block painted white is fault-free. As is shown in the figure, each 1-arrow in a Hamiltonian cycle generates at most two dirty blocks, while they may overlap with dirty blocks generated by the other 1-arrows.

In the following, we will consider the following three fault-diagnosis procedures separately: 1) for m -cubes in \mathcal{Q} with at most two dirty blocks (Case A), 2) for m -cubes with at least three and at most $m - \ell$ dirty blocks (Case B), and 3) for m -cubes with at least $m - \ell + 1$ and at most $2m$ dirty blocks (Case C).

Case A. In the proposed scheme, we use several m -cubes with at most one faulty vertex for the testing of other m -cubes. Note that an m -cube can contain exactly one faulty vertex only when either 1) it contains a single 1-arrow, or 2) it contains two consecutive 1-arrows. Note that in each of the cases, a faulty vertex generates at most two dirty blocks in an m -cube. Since $m - \ell \geq 2$ for $m \geq 5$, a dirty block in such m -cubes can be tested by other blocks contained in the same m -cube. In addition, when $m \geq 6$, we can select such a “testing block” from at least two candidates in the same m -cube, since it holds $m - \ell \geq 3$.

Case B. Next, let us focus on m -cubes that is known to contain more than one faulty vertices by the outcome of the first two rounds (i.e., m -cubes that contain at least two non-adjacent 1-arrows). Let $\mathcal{Q}' (\subseteq \mathcal{Q})$ be the set of such m -cubes. Since it can contain at most m faulty vertices, and a faulty vertex can generate at most two dirty blocks, the number of dirty blocks in an m -cube in \mathcal{Q}' is at most $2m$. Let y denote the number of dirty blocks in an m -cube in \mathcal{Q}' . If $y \leq m - \ell$, we can complete the diagnosis for the corresponding m -cube by spending one more round, in a similar way to the case of $x \leq n - m$ (note that the diagnosis process can be completed locally in each m -cube).

Case C. When $m \geq 5$, there are at most three such m -cubes in \mathcal{Q}' that contain at least $m - \ell + 1$ dirty blocks. Let $\mathcal{Q}'' (\subseteq \mathcal{Q}')$ be the set of those m -cubes. Since $|\mathcal{Q}'| \leq m - 1$, each m -cube in \mathcal{Q}'' has at least $(n - m) - (m - 2) = n - 2m + 2$ adjacent m -cubes each of which is not a member of \mathcal{Q}' (recall that an m -cube not in \mathcal{Q}' contains at most two dirty blocks); more specifically, when $n \geq 8$, there are at least six such m -cubes. Hence we can assign two such m -cubes for each member of \mathcal{Q}'' .

Let P_1 be a member of \mathcal{Q}'' and P_4 and P_5 be two m -cubes assigned to P_1 , without loss of generality. Recall again that each of P_4 and P_5 contains at most two consecutive dirty blocks. We examine the following two cases separately:

- When $m \geq 6$, by selecting “testing blocks” in P_4 and P_5 appropriately, we can prepare at least $2^{m-\ell} - 2$ fault-free blocks for testing dirty blocks in P_1 (we could not cover at most two blocks, and such a situation occurs only when the dirty blocks in P_4 and P_5 are adjacent with those in P_1). See Figure 2 for illustration.
- When $m = 5$, since $x \geq n - m + 1$ and there is an m -cube containing at least two faulty vertices, we have $m' \leq 3$. Thus, $\ell = \lceil \log_2 m' \rceil \leq 2$; i.e., $m - \ell \geq 3$. Hence the same argument to the above case holds.

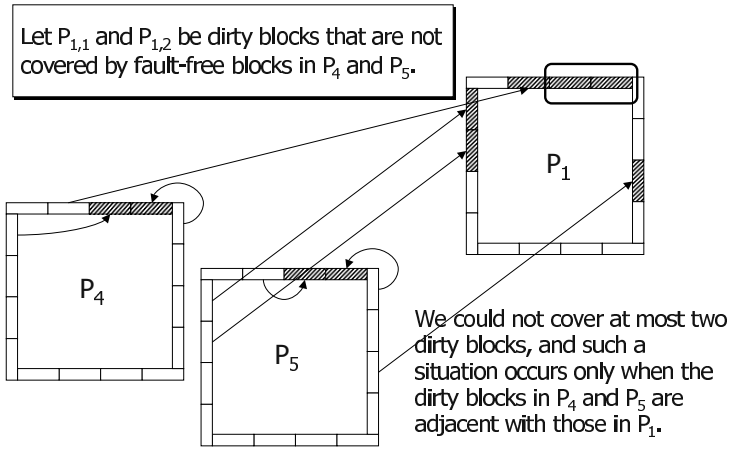


Fig. 2. Case C

Let $P_{1,1}$ and $P_{1,2}$ be dirty blocks that are not covered by fault-free blocks in P_4 and P_5 . Note that those blocks correspond to two neighboring ℓ -cubes in P_1 . If $P_{1,1}$ is adjacent with a fault-free block in P_1 , the diagnosis process completes in one more round, and the same is true for $P_{1,2}$. On the other hand, if all blocks adjacent with $P_{1,1}$ in P_1 are dirty, it implies that P_1 contains at least $m - \ell$ faulty vertices belonging to different ℓ -cubes (a faulty vertex not in $P_{1,1} \cup P_{1,2}$ does not affect to both $P_{1,1}$ and $P_{1,2}$ simultaneously, since Q_n is bipartite); i.e., the number of identified faulty vertices in the overall system is at least $(n - m + 1) + (m - \ell) = n - \ell + 1$, and this fact implies that the value of m' could be reduced to $\lceil \log_2 m' \rceil (= \ell)$.

Base Case. For the new m' , we repeat the same checking process, and the repetition continues until: 1) it finds that the diagnosis completes by spending one more round, or 2) it becomes $m' = 2$. However, if there is a block that does not adjacent with a fault-free block in P_1 for $m' = 2$, since $m' = 2$ implies $\ell = 1$, there must exist $n - \ell + 1 = n$ “nonadjacent” faulty vertices, and all of those faulty vertices could be uniquely identified.

Hence in every case, we can complete the diagnosis by spending one more round.

3.4 Small m 's

In this subsection, for completeness, we provide concrete diagnosis schemes for $m \leq 4$. Recall that $m = 4$ when $8 \leq n \leq 15$, $m = 3$ when $4 \leq n \leq 7$ and $m = 2$ when $n = 3$.

$m = 4$. Suppose again that \mathcal{Q} denotes the set of m -cubes containing at least one 1-arrow, and \mathcal{Q}' denotes the set of m -cubes that have been known to contain at least two faulty vertices. When $m = 4$, since each element in \mathcal{Q}' can contain at most 4 faulty vertices, we can initialize parameters as $m' = 4$ and $\ell = 2$; i.e.,

initially, each Hamiltonian cycle of size 16 ($= 2^4$) is partitioned into four blocks of size four each.

When $\ell = 2$, \mathcal{Q}'' denotes the set of 4-cubes containing at least 3 ($= m - \ell + 1$) dirty blocks. Without loss of generality, we may assume \mathcal{Q}'' contains either one, two, or three elements, since if $|\mathcal{Q}''| = \emptyset$, we can immediately complete the diagnosis (See Subsections 3.3 and 3.3).

- If $|\mathcal{Q}''| = 3$, all of n faulty vertices can be uniquely identified, since $|\mathcal{Q}| \geq n - m + 1 = n - 3$ and each element in \mathcal{Q}'' is known to contain at least two faulty vertices. More concretely, every vertex pointed by 1-arrow is identified as a faulty one.
- If $|\mathcal{Q}''| = 2$, the value of m' could be reduced to two since $n - 1$ faulty vertices has already been identified. For the new m' , we can directly apply the same procedure given in Subsection 3.3.
- If $|\mathcal{Q}''| = 1$, the unique element in \mathcal{Q}'' contains at least two faulty vertices, and is adjacent with $n - 4$ m -cubes in the given n -cube. Without loss of generality, we may assume that each of the $n - 4$ neighbors belongs to \mathcal{Q}' and contains at most two consecutive 1-arrows in it (since otherwise, we can directly apply the procedure given in Subsection 3.3). Hence, we can decrease ℓ to 1, and for the reduced parameter ℓ , we can directly apply the same procedure, as well.

$m = 3$. When $m = 3$, we may assume that each m -cube in \mathcal{Q}' contains at most three faulty vertices. Thus, we can set parameters as $m' = 3$ and $\ell = 2$. If \mathcal{Q}'' contains either zero, two, or three elements, a similar method to the case of $m = 4$ can be used. Thus we may assume \mathcal{Q}'' contains exactly one element. If $|\mathcal{Q}''| = 1$, the unique element in \mathcal{Q}'' contains at least two faulty vertices, and is adjacent to $n - 3$ m -cubes in the given n -cube. Without loss of generality, we may assume that each of the $n - 3$ neighbors belongs to \mathcal{Q}' and contains at most two consecutive 1-arrows in it. Hence, we can decrease ℓ to 1, and can apply the procedure given in Subsection 3.3, as well.

$m = 2$. When $m = 2$ (i.e., when $n = 3$), two directed Hamiltonian cycles with four vertices are constructed in the first two rounds. Let $A = (a_1, a_2, a_3, a_4)$ and $B = (b_1, b_2, b_3, b_4)$ be syndromes obtained in the first two rounds, where $a_i, b_i \in \{0, 1\}$ for each i (for brevity, two rotated syndromes will be regarded as an identical one).

If either A or B contains no 1-arrows, i.e., if it is $(0, 0, 0, 0)$, all vertices contained in the cycle must be fault-free, and in the third round, they can be used to test four other vertices (if necessary). Hence in the following, we assume that each syndrome contains at least one 1-arrow, i.e., each Hamiltonian cycle contains at least one and at most two faulty vertices (recall again that we are assuming that the total number of faults does not exceed three).

A Hamiltonian cycle contains exactly one faulty vertex only when the corresponding syndrome is either $(0, 0, 1, 0)$ or $(0, 0, 1, 1)$, and in each case, two vertices pointed by the first two 0-arrows must be fault-free since it can contain

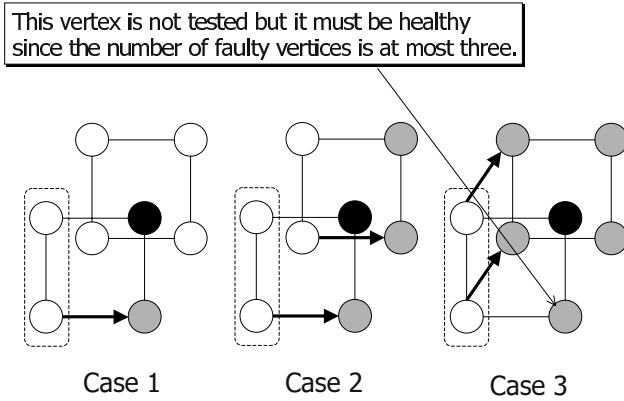


Fig. 3. Possible three test assignments placed in the third round

at most two faulty vertices. Without loss of generality, we assume that syndrome A is $(0, 0, 1, *)$, where $*$ denotes 0 or 1.

Consider the following three cases separately:

- If $B = (0, 0, 1, 0)$ or $(0, 0, 1, 1)$, we can complete the diagnosis as is shown in Case 2 in Figure 3 (b).
- Suppose B is either $(0, 1, 0, 1)$, $(0, 1, 1, 1)$, or $(1, 1, 1, 1)$. Since it must contain more than one faulty vertices, and of course, since it can contain at most two faulty vertices, the pattern of faulty vertices follows one of four possible ways of selecting two faulty vertices from the cycle. In addition, we can uniquely identify the actual fault pattern by testing the fault-freeness of only two vertices in the cycle, as is shown in Case 3 in Figure 3. It should also be worth noting that, for such B 's, the cycle corresponding to syndrome A contains exactly one fault, that results in the fault-freeness of the last vertex in the cycle, as is illustrated in the same figure.

4 Concluding Remarks

In this paper, we consider the problem of adaptive fault-diagnosis in binary n -cubes with at most n faulty vertices, and propose a scheme that completes a diagnosis in at most three test rounds, provided $n \geq 3$. The basic idea of the proposed scheme could be applied to other classes of sparse network topologies, such as star graphs and pancake graphs, that is probably the most interesting open problem. An extension of the fault model of each unit is an important direction of further research.

References

1. R. Beigel, W. Hurwood, N. Kahale, “Fault diagnosis in a flash,” in *Proc. 36th FOCS*, pp. 571–580 (1995).

2. A. Björklund, "Optimal adaptive fault diagnosis of hypercubes," in *Proc. SWAT 2000*, LNCS 1851, pp. 527–534 (2000).
3. P. M. Blecher, "On a logical problem," *Discrete Math.*, 43: 107–110 (1983).
4. C. Feng, L. N. Bhuyan, and F. Lombardi, "Adaptive system-level diagnosis for hypercube multiprocessors," *IEEE Trans. Comput.*, 45(10): 1157–1170 (Oct. 1996).
5. S. L. Hakimi, A. T. Amin, "Characterization of connection assignment of diagnosable systems," *IEEE Trans. Comput.*, C-23(1): 86–88 (Jan. 1974).
6. S. L. Hakimi, K. Nakajima, "On adaptive system diagnosis," *IEEE Trans. Comput.*, C-33(3): 234–240 (March 1984).
7. E. Kranakis, A. Pelc, and A. Spatharis, "Optimal adaptive diagnosis for simple multiprocessor systems," *Networks*, 34: 206–214 (1999).
8. E. Kranakis and A. Pelc, "Better adaptive diagnosis of hypercubes," *IEEE Trans. Comput.*, 49(19): 1013–1020 (Oct. 2000).
9. K. Nakajima, "A new approach to system diagnosis," in *Proc. 19th Allerton Conf. Commun. Control and Computing*, pp. 697–706 (1981).
10. K. Nomura, T. Yoshida, S. Ueno, "On adaptive fault diagnosis for multiprocessor systems," in *Proc. ISAAC 2001*, LNCS 2223, pp. 86–98 (2001).
11. A. Okashita, T. Araki, Y. Shibata, "An optimal adaptive diagnosis of butterfly networks," *IEICE Trans. Fundamentals*, E86-A(5): 1008–1018 (May 2003).
12. F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Trans. Electron. Comput.*, EC-16(6): 848–854 (Dec. 1967).

Fast Algorithms for Comparison of Similar Unordered Trees

Daiji Fukagawa¹ and Tatsuya Akutsu²

¹ Department of Intelligence Science and Technology, Graduate School of Informatics, Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501 Japan

² Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011 Japan
{daiji, takutsu}@kuicr.kyoto-u.ac.jp

Abstract. We present fast algorithms for computing the largest common subtree (LCST) and the optimal alignment when two similar unordered trees are given. We present an $O(4^K n)$ time algorithm for the LCST problem for rooted trees, where n is the maximum size of two input trees and K is the minimum number of edit operations to obtain LCST. We extend this algorithm to unrooted trees and obtain an $O(K4^K n)$ time algorithm. We also show that the alignment problem for rooted and unordered trees of bounded degree can be solved in linear time if K is bounded by a constant.

1 Introduction

The problem of comparison of two labeled trees occurs in several diverse areas such as computational biology, computational chemistry, image recognition, and structured text databases. Therefore, extensive studies have been done on this.

In particular, many studies have been done on computing the edit distance between two ordered trees [7, 9, 14, 18] because this is useful in comparing RNA secondary structures. Tai [14] defined the edit distance between two ordered trees and developed an $O(n^2 D^4)$ time algorithm, where n denotes the maximum size (i.e., maximum number of nodes) of two input trees and D denotes their maximum depth. Shasha and Zhang [18] developed an improved $O(n^2 \min\{L, D\}^2)$ time algorithm (where L denote the number of leaves) and Klein [9] further developed an $O(n^3 \log n)$ time algorithm. Jiang et al. considered the problem of computing the optimal alignment of two trees [8] and developed an $O(n^2 \Delta^2)$ time algorithm for two ordered trees, where Δ denotes the maximum number of children of any node in two input trees.

Although many existing studies have focused on ordered trees, comparing unordered trees is also important because chemical structures are usually unordered. Although chemical structures are not necessarily trees, algorithms for tree-like structures have been developed [2, 16] based on algorithms for com-

paring unordered trees. Furthermore, glycans have unordered and rooted tree structures in general, which conform to a special but important family of chemical compounds [4]. In comparing unordered trees, the problem of finding the largest common subtrees (LCST) is important. The LCST between two unordered trees can be computed in $O(n^3)$ time [11, 15]. If the maximum degree is bounded by a constant, it works in $O(n^2)$ time. Shamir and Tsur [12] developed an $O(n^{2.5}/\log n)$ time algorithm for a closely related problem (the subtree isomorphism problem). Several hardness results are also known for comparing unordered trees. Zhang and Jiang [17] proved that the edit distance problem is MAX SNP-hard for unordered trees (even if $\Delta = 2$) whereas Zhang et al. [19] developed an $O(n^2)$ algorithm when the number of leaves is constant. Jiang et al. [8] proved that the alignment problem is MAX SNP-hard in general whereas it can be solved in polynomial time if the maximum degree is bounded by a constant. Akutsu and Halldórsson [3] proved that the LCST problem for many trees is very hard to approximate.

Here, we consider special cases in which there are at most K differences between two input trees, inspired from $O(nK)$ time algorithms developed for approximate string matching [10]. Our objective is to develop linear time algorithms for small K , which may decrease the gaps between the linear time algorithm for tree isomorphism [1] and the algorithm for LCST [11, 15] and other algorithms for comparing trees. For these special cases, Shasha and Zhang [13] developed an $O(K^2nH)$ time algorithm for the unit cost edit distance problem for ordered trees where H is the minimum height of two input trees, and Jansson and Lingas [7] developed an $O(n \log n \Delta^4 K^2)$ time algorithm for the alignment problem for ordered trees. However, it seems difficult to extend these algorithms to unordered trees because both are based on some properties such as the left-to-right post order numbers of corresponding nodes cannot differ very much. In comparing unordered trees, we cannot use these properties. Indeed, Jansson and Lingas wrote in [7] that it is an interesting open problem whether substantial speed-up in the construction of an optimal alignment between similar unordered trees of bounded degree is achievable.

In this paper, we present an $O(4^K n)$ time algorithm for the LCST problem for two rooted and unordered trees. It should be noted that this algorithm works in linear time for constant K and in subquadratic time for $K = o(\log n)$. We extend this algorithm to undirected trees and obtain an $O(K(4^K n))$ time algorithm. We show that the alignment problem with unit cost editing operations for two unordered binary trees can be solved in $O(4.45^K n)$ time, which can be extended to a linear time algorithm for constant K and unordered trees of bounded degree. This result partially answers the question posed by Jansson and Lingas. Although the proposed algorithms are not practical, the results are non-trivial and novel techniques are introduced. Almost all existing algorithms for comparing trees are based on dynamic programming and thus employ a bottom-up approach, whereas our proposed algorithms employ a top-down approach. We also present faster algorithms for special cases of the LCST problem.

2 Preliminaries

For a graph G , we denote the set of nodes and edges by $V(G)$ and $E(G)$ respectively. Let T be a rooted tree. The root of T is denoted by $\text{root}(T)$. The size of T , denoted by $|T|$, is defined as $|V(T)|$. The depth of a node $v \in V(T)$, $\text{depth}(v)$, is the number of edges on the path from v to $\text{root}(T)$, therefore $\text{depth}(\text{root}(T)) = 0$. The children of a node v , $\text{children}(v)$, is the number of children of v .

For a node v in T , we write $T[v]$ to denote the subtree of T rooted at v . Therefore, $T[v]$ includes v and all nodes in T that are descendants of v . If T is a unordered tree, we use $T[U]$ to denote the subforest of T that consists of subtrees rooted at any node in U , where U is a subset of $V(T)$ and any node in U is not a descendant of the other node in U .

For unrooted trees, we can consider the parental relation between edges, as well as nodes in the rooted case. For a pair of directed edges $(u, v), (v, w) \in E(T)$ such that $u \neq w$, (v, w) is called a child of (u, v) and (u, v) is called a parent of (v, w) . We can define a subtree of T , denoted by $T[(u, v)]$, as the rooted subtree including all edges descended from (u, v) where $u \notin V(T[(u, v)])$.

2.1 Signatures for Rooted Trees

A linear time algorithm is known for testing the isomorphism of two rooted trees [1]. It assigns an integer to each node in a bottom-up way so that the same integer number is assigned to two nodes iff subtrees induced by the nodes and their descendants are isomorphic.

Let T be an unlabeled rooted tree. We define the level of node $v \in V(T)$, denoted by $\text{level}(v)$, as follows: (i) $\text{level}(v) = 0$ if v is a leaf, and (ii) $\text{level}(v)$ is the smallest integer that is exactly larger than the level of any child of v . Note that the level of a node may be different from that of another node at the same depth in our definition. We write $\text{level}(T)$ to refer to the maximum level of any node v in T . The level of a forest F is the maximum level of any trees in F .

Definition 1. Let $F = \{T_1, \dots, T_p\}$ be a forest. For each node $v \in V(F)$, we define the signature $\sigma(v)$ of v as follows: $\sigma(v) = 0$ if v is a leaf, and $\sigma(v) = \max_{u \in \text{children}(v)} \sigma(u) + 1$ otherwise. We write $\sigma(F)$ to refer to the maximum signature of any node v in F .

Lemma 1. Let F be a forest. For each node $v \in V(F)$, we define the level of v as follows: $\text{level}(v) = \sigma(v)$ if v is a leaf, and $\text{level}(v) = \max_{u \in \text{children}(v)} \text{level}(u) + 1$ otherwise.

Since we adopted the random access machine (RAM) model as used in many other studies, we can assume that any integer of $O(\log N)$ bits can be stored in one word and accessed in a unit time where N is the length of the input.

We assign an integer to each node in F as follows. First, we assign 0 to each leaf, i.e., each node at level 0. At each step h , we assign an integer to node v if

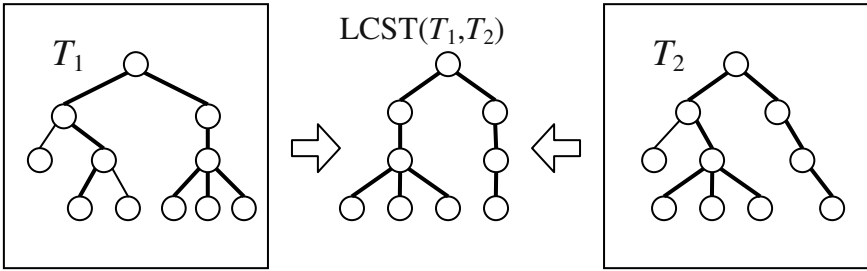


Fig. 1. Example of largest common subtree (LCST). The LCST between T_1 and T_2 is indicated by thick edges. In this case, $\hat{d}(T_1, T_2) = 3$

$(v) = h$ according to the signatures of v 's children. It should be noted that there is a one-to-one correspondence between nodes and subtrees, hence at most n integers suffice where n is the number of nodes in F . \square

2.2 Problems

Various distance measures for trees have been proposed to compute similarity between trees. In this paper, we will consider the following distance measures.

Definition 2. Difference $\hat{d}(T_1, T_2)$ is the minimum number of nodes that must be deleted from T_1 and T_2 to make them identical.

$\hat{d}(T_1, T_2)$ is a special case of the edit distance of trees, except for the difference that the edit distance admits the contraction of internal nodes. In particular, we write $\hat{d}_K(T_1, T_2)$ to denote $\hat{d}(T_1, T_2)$ within K differences, i.e., $\hat{d}_K(T_1, T_2)$ is equal to $\hat{d}(T_1, T_2)$ if $\hat{d}(T_1, T_2) \leq K$ and $+\infty$ otherwise.

Definition 3 (Jiang et al., [8]). An alignment between trees T_1 and T_2 is a pair of trees (A_1, A_2) such that A_1 is a subtree of T_1 and A_2 is a subtree of T_2 , and A_1 and A_2 are isomorphic. The cost of an alignment is the number of nodes in T_1 and T_2 that are not in A_1 and A_2 respectively. An optimal alignment is an alignment with minimum cost. The alignment distance between T_1 and T_2 is the cost of an optimal alignment.

Note that we only consider unlabeled trees and define the cost of alignment as the number of spaces although (\dots) . We refer to the alignment distance between T_1 and T_2 as $\tilde{d}(T_1, T_2)$.

For each distance measure, $d_{\leq K}(T_1, T_2)$ denotes $d(T_1, T_2)$ if it is not larger than K and ∞ otherwise. Each of these measures is a distance metric, since

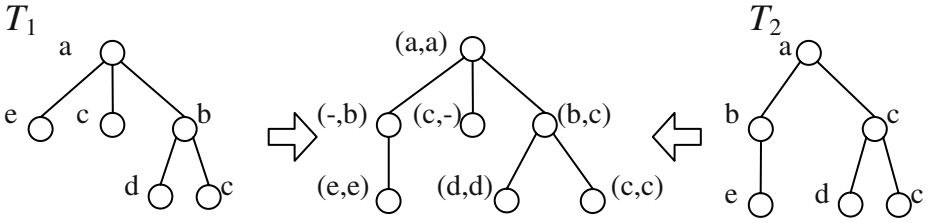


Fig. 2. Example of tree alignment with labeled nodes. Here, $\tilde{d}(T_1, T_2) = 3$

the following requirement for distances are clearly satisfied: (i) isolation, (ii) symmetry, and (iii) triangular inequality.

We define the k -differences problem for trees:

LCST-kDIFF (Largest Common Subtree with K -Differences). Given two trees, T_1 and T_2 , and an integer K , output $\hat{d}_{\leq K}(T_1, T_2)$.

We call this problem the LCST-kDIFF (largest common subtree problem) with k differences because it is equivalent to the LCST problem under the restriction that there is a common subtree with a size of at least $\frac{|T_1|+|T_2|-K}{2}$ between T_1 and T_2 .

LCST-kDIFF is a subproblem of the LCST, therefore it can be solved in polynomial time [11, 2]. While there is no linear time algorithm for LCST, our algorithm for LCST-kDIFF runs in linear time if both K and Δ are bounded by fixed constants. We define the k -differences problem for tree alignment in the same way.

TALI-kDIFF (Tree Alignment Within K -Distance). Given two trees T_1, T_2 and an integer K , output the alignment distance $\tilde{d}_{\leq K}(T_1, T_2)$.

3 Largest Common Subtree with K Differences

For two trees, we can easily see that LCST-kDIFF can be solved in at most $O(n^3)$ time ($O(n^2)$ time if the degree is bounded) by computing the LCST between them [2, 4, 11, 12, 15].

For rooted trees, let $R[u, w]$ denote the size of the LCST between $T_1[u]$ and $T_2[w]$. We can compute R 's with the dynamic programming technique:

$$R[u, \emptyset] = R[\emptyset, w] = 0, \quad R[u, w] = 1 + \max_{\psi \in \mathcal{M}(u, w)} \left\{ \sum_{parent(u_i)=u} R[u_i, \psi(u_i)] \right\},$$

where \mathcal{M} denotes the set of mappings from the set of children of u to the set of children of v .

Although there are an exponentially large number of mappings, the maximum matching can be computed in polynomial time. The combination of maximum matching and dynamic programming can also be found in a study on the symmetry number problem [5].

Algorithm LCST-KDIFF**Input:** $T_1[u], T_2[w], k$ **Output:** $\hat{d} \leq k(T_1[u], T_2[w])$ Assume w.l.o.g. that $\text{deg}(u) \leq \text{deg}(w)$. $U, W \leftarrow u$'s and w 's children.Sort U and W in ascending order of σ . $i \leftarrow 1, j \leftarrow 1, U' \leftarrow \emptyset, W' \leftarrow \emptyset$ **while** $i \leq |U|$ and $j \leq |W|$ **do** **if** $\sigma(u_i) = \sigma(w_j)$ **then** $i \leftarrow i + 1, j \leftarrow j + 1$ **elseif** $\sigma(u_i) < \sigma(w_j)$ **then** $U' \leftarrow U' \cup \{u_i\}, i \leftarrow i + 1$ **else** $W' \leftarrow W' \cup \{w_j\}, j \leftarrow j + 1$ **end****end** $U' \leftarrow U' \cup \{u_i, \dots, u_{|U|}\}, W' \leftarrow W' \cup \{w_j, \dots, w_{|W|}\}$ **if** $||T_1[U']| - |T_2[W']|| > k$ or $|W'| > k$ **then** output ∞ **end****if** $|U'| = 0$ **then** output $|T_2[W']|$ **end****if** $|U'| = |W'| = 1$ **then** output LCST-KDIFF($T_1[u_i], T_2[w_j], k$) **end**
($u_i \in U', w_j \in W'$)**for** each pair $u_i \in U'$ and $w_j \in W'$ **do** **if** $\sigma(u_i) = \sigma(w_j)$ **then** $\text{diff}(u_i, w_j) \leftarrow 0$ **end** $\text{diff}(u_i, w_j) \leftarrow \text{LCST-KDIFF}(T_1[u_i], T_2[w_j], k - |W'| + 1)$ **end**Find minimum cost assignment α from U' to W' using cost function diff .Output $\sum_{u_i \in U'} \text{diff}(u_i, \alpha(u_i)) + \sum_{w_j \in W' \setminus \alpha(U')} |T_2[w_j]|$ if it is not larger than k , otherwise output ∞ .**Fig. 3.** Algorithm for the k differences problem for unordered and rooted trees**3.1 Rooted Case**

First, let us consider a rooted case for the problem.

We give here a fast algorithm for the unlabeled and rooted case of LCST-KDIFF (Fig. 3). We consider that all trees are unlabeled because our algorithm can be easily extended to labeled cases.

The correctness of the algorithm follows from the following lemma.

Lemma 2. Let u_1, u_2, \dots, u_p be the children of u in T_1 and w_1, w_2, \dots, w_q be the children of w in T_2 . Let $\sigma(u_i) = \sigma(w_j)$.

We can match an arbitrary pair (u_i, w_j) with cost 0 (i.e., $\sigma(u_i) = \sigma(w_j)$) because the triangular inequality holds for $\hat{d}(u_i, w_j)$. \square

Executing $\text{LCST-KDIFF}(T_1[\dots(T_1)], T_2[\dots(T_2)], K)$, we can compute distance $\hat{d}_{\leq K}(T_1, T_2)$. Fig. 4 is an example of executing LCST-KDIFF for two rooted trees. Now we will analyse the time complexity of our algorithm. $\mathcal{R} = \langle u_1, w_1, k_1 \rangle, \langle u_2, w_2, k_2 \rangle, \dots$ denotes the sequence of all recurrence calls that execute the “for” loop (we refer to them as \dots) in computing $\text{LCST-KDIFF}(T_1[\dots(T_1)], T_2[\dots(T_2)], K)$.

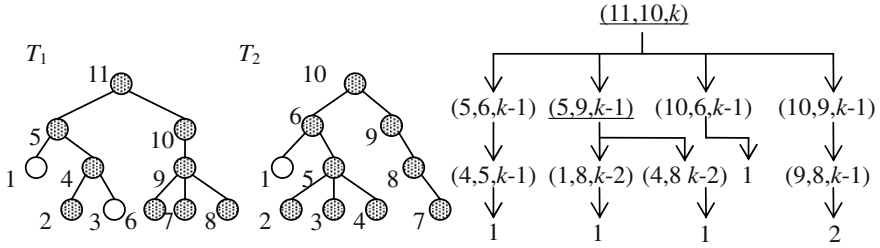


Fig. 4. Example of execution of algorithm LCST-kDIFF. LCST is indicated by shaded nodes. In figure at right, each node denotes recursive call, and nondegenerate calls are underlined

Proposition 1. For any $u \in V(T_1), w \in V(T_2)$, $\text{LCST-kDIFF}(T_1[u], T_2[w], k)$

Note that each nondegenerate recurrence call with some u, w and k can directly call LCST-kDIFF with $k - |W'| + 1 (< k)$ at most $\min\{|W'|^2, k^2\}$ times. Therefore, we can easily see that LCST-kDIFF with k can call LCST-kDIFF with $k - 1$ at most 4 times, and also call (directly and/or indirectly) LCST-kDIFF with $k - 2$ at most 4^2 times, and so on.

Proposition 2. $k \leq K \implies \mathcal{R} \leq 4^{K-k} \implies \mathcal{R} = \sum_{k=1}^K 4^{K-k} = O(4^K)$

Theorem 1. $\text{LCST-kDIFF}(T_1, T_2) = O(4^K n) = O(n \hat{d}(T_1, T_2)^K)$

We need linear time preprocessing to compute the signatures.

First, we consider the latter part of the algorithm (computing “for” loop and minimum cost assignment). This part of each recursive call $\langle u, w, k \rangle$ excepting the inside of recursive calls can be computed in polynomial time regarding k . Using Prop. 2, the total time cost for this part is $\sum_{k=0}^K 4^{K-k} = O(4^K)$.

Then, we prove that computation of the remaining part can be done in $O(4^K n)$ time. Let us consider a list of recursive calls $\langle u_0, w_0, k_0 \rangle, \dots, \langle u_p, w_p, k_p \rangle$ such that $\langle u_i, w_i, k_i \rangle$ is called by $\langle u_{i-1}, w_{i-1}, k_{i-1} \rangle$ for each $1 \leq i \leq p$, there are no recursive calls called by $\langle u_p, w_p, k_p \rangle$ and $u_0 = \dots(T_1), w_0 = \dots(T_2), k_0 = K$. The total time cost in computing all recursive calls in this list except the latter part of the algorithm is $\sum_{i=0}^p O(|u_i| + |w_i|) = O(n)$. Note that both sorting U and W and removing isomorphic pairs can be done in $O(|u| + |w|)$ time. Using Prop. 2, the number of such lists in \mathcal{R} is at most 4^{K-1} . Thus, the time cost for the remaining part is at most $O(4^K n)$.

Hence, the algorithm takes at most $O(4^K n)$ time in total. □

Corollary 1. $\text{LCST-kDIFF}(T_1, T_2) = O(n \hat{d}(T_1, T_2)^K) = O(n \log^K n)$

3.2 Unrooted Case

Now, we extend LCST-kDIFF to unrooted trees.

Theorem 2. *Let T_1 and T_2 be two unordered trees with n nodes. Then, LCST can be solved in $O(K4^K n)$ time.*

We can assume K is sufficiently small (e.g., $K = o(n)$), otherwise $O(4^K n)$ time can be achieved with the existing algorithm [11, 15]. Since LCST-kDIFF can be solved in $O(4^K n)$ time once the roots of T_1 and T_2 are known, we will show how to search the root pair.

We prove that there is a node in T_1 such that it is included in LCST and it corresponds to at most $O(K)$ nodes in T_2 .

Let $u \in T_1$ be a node such that $|T_1[(u, w)]| \leq \frac{n}{2}$ holds for any w adjacent to u . We can easily see that any tree T_1 has either one such node u , or two adjacent nodes u, u' for which $|T_1[(u, u')]| = |T_1[(u', u)]| = \frac{n}{2}$ holds. Note that u is included in LCST as for we assumed that K is small. Let u (either of u and u' in the latter case) correspond to the root of LCST.

For $k = 0, 1, \dots, K$, let C_k be the set of nodes such that $|T_2[(u, w)]| \leq \frac{n}{2} + k$ holds for any $u \in C_k$ and any w adjacent to u . Clearly, $C_0 \subseteq C_1 \subseteq \dots \subseteq C_K \subseteq V(T_2)$. Furthermore, C_k construct a path in T_2 and $|C_k \setminus C_{k-1}| \leq 2$ for any $1 \leq k \leq K$ if K is sufficiently small.

Let us consider another tree T' such that $\hat{d}(T', T_2) = 1$ holds. Similarly, let C'_k ($k = 0, 1, \dots, K$) be the set of nodes such that $|T'[(u, w)]| \leq \frac{n}{2} + k$ holds for any $u \in C'_k$ and any w adjacent to u . It is obvious that the set of nodes corresponding to C_k is included in C'_{k+1} (and C'_k is included in the set of nodes corresponding to C_{k+1}) since only one node differs between them. By repeating this, we can see that the node corresponding to the root of LCST is included in $C'_K \subseteq V(T_2)$ if $\hat{d}(T_1, T_2) \leq K$. Therefore, we need to examine at most $O(K)$ pairs, and the theorem follows. \square

4 Faster Algorithms for Special Cases of LCST

4.1 Rooted Ordered Case

LCST-kDIFF for two rooted ordered trees can easily be solved with dynamic programming using the technique proposed in [13].

Let $id_1(u)$ (resp. $id_2(w)$) denote the left-to-right postorder number of node u in T_1 (resp. node w in T_2). The following lemma can be proven as in [13].

Lemma 3. *Let $u \in T_1$ and $w \in T_2$ be two nodes. Then, $|id_1(u) - id_2(w)| \leq K$.*

For each pair of nodes (u, w) such that $|id_1(u) - id_2(w)| \leq K$, we compute $\hat{d}(T_1[u], T_2[w])$ in a bottom-up manner where we also use $\hat{d}(T_1, T_2)$ for rooted ordered trees. Suppose that u has children u_1, u_2, \dots, u_p and w has children w_1, w_2, \dots, w_q , respectively. We compute $\hat{d}(T_1[u], T_2[w])$ by using the following DP procedure, where $\hat{d}(T_1[u], T_2[w]) = D(p, q)$ (for unlabeled trees).

$$D(0, 0) := 0, \quad D(i, 0) := \sum_{i' \leq i} |T_1[u_{i'}]|, \quad D(0, j) := \sum_{j' \leq j} |T_2[w_{j'}]|,$$

$$D(i, j) := \min \begin{cases} D(i-1, j) + |T_1[u_i]|, \\ D(i, j-1) + |T_2[w_j]|, \\ D(i-1, j-1) + \hat{d}(T_1[u_i], T_2[w_j]) \end{cases}$$

Since $D(p, q)$ must be at most K , we only need to compute $D(p, q)$ for (p, q) 's satisfying $|p - q| \leq K$.

Theorem 3. *Let T_1 and T_2 be rooted ordered trees with n nodes. Then, LCST can be computed in $O(K^2n)$ time.*

Since we only need to compute $D(p, q)$ for (u, w) such that $|deg(u) - deg(w)| \leq K$, computation of $D(p, q)$ takes $O(deg(u)K)$ time per (u, w) . For each u , we need to compute $\hat{d}(T_1[u], T_2[w])$ for at most $2K$ nodes w . Therefore, the total time is

$$\sum_{u \in T_1} O(deg(u)K^2) = K^2 \sum_{u \in T_1} O(deg(u)) = O(K^2n).$$

□

4.2 When Siblings Have Distinct Labels

Here, we consider a special case of rooted ordered trees, in which all children of each node have distinct labels. This special case is often considered in tree pattern matching [6].

Let r_1 and r_2 be the roots of the subtrees of T_1 and T_2 , each of which corresponds to the root of LCST. We assume w.l.o.g. that the size of LCST is greater than $\frac{3n}{4}$ (i.e., $K < \frac{n}{4}$) because we are going to show an $O(Kn)$ time algorithm.

Proposition 3. *Let r_1 and r_2 be the roots of the subtrees of T_1 and T_2 that correspond to the root of LCST. Then, the number of nodes in the subtree rooted at r_1 (resp. r_2) is at least $\frac{3n}{4} - K$.*

Let u_1, \dots, u_d be the children of an arbitrary node u . Then, $|T_1[u_i]| \geq \frac{3n}{4}$ holds at most one node u_i . □

If we know the pair of the roots (r_1, r_2) , LCST can be computed in linear time by traversing both trees simultaneously. It follows from the above proposition that LCST can be computed in $O(K^2n)$ time. However, we can further reduce the computation time. Let P_1 (resp. P_2) be the path for T_1 (resp. T_2) in the proposition.

Proposition 4. *Let $u \in P_1$ and $w \in P_2$ be the nodes of LCST. Then, there exist $u' \in P_1$ and $w' \in P_2$ such that $|T_1[u']| \geq \frac{3n}{4}$ and $|T_2[w']| \geq \frac{3n}{4}$.*

Since $|T_1[u']| \geq \frac{3n}{4}$ and $|T_2[w']| \geq \frac{3n}{4}$, u' must correspond to w' in order to have an LCST whose size is larger than $\frac{3n}{4}$. □

Algorithm FORESTALIGN-KDIFF-BIN

Input: $T_1[u], T_2[w], k$

Output: $\tilde{d}_{\leq k}(T_1[\{u_1, u_2\}], T_2[\{w_1, w_2\}])$

if $k < 0$ then output ∞ end

$$k' \leftarrow \min_{i=1,2} \begin{cases} \text{FORESTALIGN-KDIFF-BIN}(T_1[u_i], T_2[w], k - |T_1[u_{3-i}]] - 1) \\ \quad + |T_2[u]| - |T_2[u_i]|, \\ \text{FORESTALIGN-KDIFF-BIN}(T_1[u], T_2[w_i], k - |T_2[w_{3-i}]] - 1) \\ \quad + |T_2[w]| - |T_2[w_i]|, \\ \text{TREEALIGN-KDIFF-BIN}(T_1[u_1], T_2[w_i], k' - 1) \\ \quad + \text{TREEALIGN-KDIFF-BIN}(T_1[u_2], T_2[w_{3-i}], k' - 1) \end{cases}$$

Output k' if $k' \leq k$, otherwise output ∞ .

Algorithm TREEALIGN-KDIFF-BIN

Input: $T_1[u], T_2[w], k$

Output: $\tilde{d}_{\leq k}(T_1[u], T_2[w])$

if $\sigma(u) = \sigma(w)$ and $k \geq 0$ then output 0 end

if $k \leq 0$ then output ∞ end

if there exists a pair (i, j) such that $\sigma(u_i) = \sigma(w_j)$ then

output TREEALIGN-KDIFF-BIN($T_1[u_{3-i}], T_2[w_{3-j}], k$) end

$$k' \leftarrow \min \begin{cases} \text{FORESTALIGN-KDIFF-BIN}(T_1[u], T_2[w], k), \\ \min_{i=1,2} \begin{cases} \text{TREEALIGN-KDIFF-BIN}(T_1[u_i], T_2[w], k - |T_1[u_{3-i}]] - 1) \\ \quad + |T_1[u]| - |T_1[u_i]|, \\ \text{TREEALIGN-KDIFF-BIN}(T_1[u], T_2[w_i], k - |T_2[w_{3-i}]] - 1) \\ \quad + |T_2[w]| - |T_2[w_i]| \end{cases} \end{cases}$$

Output k' if $k \leq k'$, otherwise output ∞ .

Fig. 5. Algorithm for TALI-kDIFF for unordered binary trees

Using this lemma, we can prove the following theorem as in Theorem 1.

Theorem 5. $\tilde{d}_{\leq K}(T_1, T_2) = O(4.45^K n)$

We can extend our technique to any constant Δ although the proof has been omitted.

Theorem 6. $\tilde{d}_{\leq K}(T_1, T_2) = O(K \cdot \dots T_1 \dots T_2)$

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Addison-Wesley (1974)
2. Akutsu, T.: A Polynomial Time Algorithm for Finding a Largest Common Subgraph of Almost Trees of Bounded Degree. IEICE Trans. on Information and Systems, E76-A (1992) 1488–1493
3. Akutsu, T., Halldórsson, M.M.: On the approximation of largest common subtrees and largest common point sets. Theoretical Computer Science **233** (2000) 33–50

4. Aoki, K.F., Yamaguchi, A., Okuno, Y., Akutsu, T., Ueda, N., Kanehisa, M., Mamitsuka, H.: Efficient tree-matching methods for accurate carbohydrate database queries. In: Genome Informatics. Volume 14. (2003) 134–143
5. Chin, K., Yen H.: The symmetry number problem for trees. Information Processing Letters **79** (2001) 73–79
6. Cole, R., Hariharan R.: Tree pattern matching to subset matching in linear time. SIAM J. Computing **32** (2003) 1056–1066
7. Jansson, J., Lingas, A.: A fast algorithm for optimal alignment between similar ordered trees. Fundamenta Informaticae **56** (2003) 105–120
8. Jiang, T., Wang, L., Zhang, K.: Alignment of trees — an alternative to tree edit. Theoretical Computer Science **143** (1995) 137–148
9. Klein, P.N.: Computing the edit-distance between unrooted ordered trees. In Bilardi, G., Italiano, G.F., Pietracaprina, A., Pucci, G., eds.: ESA. Volume 1461 of Lecture Notes in Computer Science., Springer (1998) 91–102
10. Landau, G.M., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms **10** (1989) 157–169
11. Matula, D.W.: Subtree isomorphism in $O(n^{5/2})$. In Alspach, B., Hell, P., Miller, D.J., eds.: Algorithmic Aspects of Combinatorics. Volume 2 of Ann. Discrete Math. North-Holland (1978) 91–106
12. Shamir, R., Tsur, D.: Faster subtree isomorphism. J. Algorithms **33** (1999) 267–280
13. Shasha, D., Zhang, K.: Fast algorithms for the unit cost editing distance between trees. J. Algorithms **11** (1990) 581–621
14. Tai, K.C.: The tree-to-tree correction problem. J. ACM **26** (1979) 422–433
15. Valiente, G.: Algorithms on trees and graphs. Springer (2002)
16. Yamaguchi, A., Mamitsuka, H.: Finding the maximum common subgraph of a partial k -tree and a graph with a polynomially bounded number of spanning trees. In Ibaraki, T., Katoh, N., Ono, H., eds.: ISAAC. Volume 2906 of Lecture Notes in Computer Science., Springer (2003)
17. Zhang, K., Jiang, T.: Some max snp-hard results concerning unordered labeled trees. Information Processing Letters **49** (1994) 249–254
18. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Computing **18** (1989) 1245–1262
19. Zhang, K., Statman, R., Shasha, D.: On the Editing Distance Between Unordered Labeled Trees. Information Processing Letters **42** (1992) 133–139

GCD of Random Linear Forms

Joachim von zur Gathen¹ and Igor E. Shparlinski²

¹ Fakultät für Elektrotechnik, Informatik und Mathematik,
Universität Paderborn,
33095 Paderborn, Germany
gathen@upb.de

<http://www-math.upb.de/~aggathen>

² Department of Computing, Macquarie University,
NSW 2109, Australia
igor@comp.mq.edu.au
<http://www.comp.mq.edu.au/~igor>

Abstract. We show that for arbitrary positive integers a_1, \dots, a_m , with probability at least $6/\pi^2 + o(1)$, the gcd of two linear combinations of these integers with rather small random integer coefficients coincides with $\gcd(a_1, \dots, a_m)$. This naturally leads to a probabilistic algorithm for computing the gcd of several integers, with probability at least $6/\pi^2 + o(1)$, via just one gcd of two numbers with about the same size as the initial data (namely the above linear combinations). Naturally, this algorithm can be repeated to achieve any desired confidence level.

1 Introduction

For a vector $\mathbf{u} = (u_1, \dots, u_m) \in \mathbb{R}^m$ we define its h -norm as

$$h(\mathbf{u}) = \max_{i=1, \dots, m} |u_i|.$$

We let $\mathbf{a} = (a_1, \dots, a_m) \in \mathbb{N}^m$ be a vector of $m \geq 2$ positive integers, $\mathbf{x} = (x_1, \dots, x_m), \mathbf{y} = (y_1, \dots, y_m) \in \mathbb{N}^m$ be two integer vectors of the same length, where $\mathbb{N} = \{1, 2, \dots\}$, and consider the linear combinations

$$\mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^m a_i x_i \quad \text{and} \quad \mathbf{a} \cdot \mathbf{y} = \sum_{i=1}^m a_i y_i.$$

Then clearly $\gcd(a_1, \dots, a_m)$ divides $\gcd(\mathbf{a} \cdot \mathbf{x}, \mathbf{a} \cdot \mathbf{y})$, and we want to show that in fact, equality holds quite often.

For an integer M , we denote by $\rho_{\mathbf{a}}(M)$ the probability that, for \mathbf{x}, \mathbf{y} chosen uniformly in \mathbb{N}^m with height at most M ,

$$\gcd(a_1, \dots, a_m) = \gcd(\mathbf{a} \cdot \mathbf{x}, \mathbf{a} \cdot \mathbf{y}). \quad (1)$$

Assuming that $\mathbf{a} \cdot \mathbf{x}$ and $\mathbf{a} \cdot \mathbf{y}$ behave as independent random integer multiples of $\gcd(a_1, \dots, a_m)$, it is reasonable to expect that (1) holds with probability

$\zeta(2)^{-1} = 6/\pi^2$ where $\zeta(s)$ is the Riemann zeta function. Here we obtain a lower bound for $\rho_{\mathbf{a}}(M)$ which for a very wide range of m, M , and $N = h(\mathbf{a})$ shows that this quantity is asymptotically at least that big. The range in which this is established improves quite substantially the corresponding result of [2]. In particular, our result implies that one can choose M of order $\ln N$ in the algorithm of [2] rather than of order N as in Corollary 3 of [2], thus reducing quite dramatically the size of the operands which arise in the algorithm of [2].

The lower bound on $\rho_{\mathbf{a}}(M)$ plays a crucial role in the analysis of a fast probabilistic algorithm for computing the gcd of several integers which has been studied in [2]. This algorithm, for any $\delta > 0$, requires only about

$$\frac{1}{\ln(\pi^2/(\pi^2 - 6))} \ln \delta^{-1} = 1.06802 \dots \ln \delta^{-1} \tag{2}$$

pairwise gcd computations, to achieve success probability at least $1 - \delta$ (where $\ln z$ is the natural logarithm of $z > 0$). For comparison, it is noted that the naive deterministic approach may require up to $m - 1$ gcd computations. A drawback of the algorithm of [2] is that for its proof of correctness to work, the arguments given to the gcd computations have to be substantially larger than the original inputs. Our results now imply that one may choose the operands of that algorithm of approximately the same size as the inputs. An exact cost analysis depends on the cost of the particular gcd algorithm, a variety of which can be found in [3].

A well-known fact says that $\gcd(a_1, \dots, a_m)$ equals 1 with probability $\zeta^{-1}(m)$ for random integers a_1, \dots, a_m ; see [4], Theorem 332, for a precise formulation in the case $m = 2$. It is important to not confuse our result which holds for arbitrary (“worst-case”) inputs with the “average-case” result which follows from this fact.

2 Main Result

We show that for a wide choice of parameters $\rho_{\mathbf{a}}(M) \geq 0.607$. More precisely, we have the following.

Theorem 1. *Let $\mathbf{a} \in \mathbb{Z}^m$ and $N = h(\mathbf{a}) > 0$. For $M > m$,*

$$\rho_{\mathbf{a}}(M) \geq \zeta(2)^{-1} - \Delta, \quad \Delta = O(\ln^{-1}(M/m) + M^{-1} \ln(MN))$$

Without loss of generality we can assume that M/m is large enough because otherwise the result is trivial. As in [2], we remark that it is enough to consider only the case $\gcd(a_1, \dots, a_m) = 1$.

We define Q as the largest integer with the condition

$$\prod_{p \leq Q} p \leq (M/m)^{1/2},$$

where the product is taken over all primes $p \leq Q$. By the Prime Number Theorem, see Theorem 4.4 of [1], we have $Q = (1/2 + o(1)) \ln(M/m)$.

Let \mathcal{L} be the set of all pairs of integer vectors $\mathbf{x}, \mathbf{y} \in \mathbb{N}^m$ with $h(\mathbf{x}), h(\mathbf{y}) \leq M$. For an integer $k \geq 2$, we denote by $P(k)$ the largest prime divisor of k , and set $P(1) = 1$. We define the following subsets:

- $\mathcal{Q} = \{(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \mid Q \geq P(\gcd(\mathbf{a} \cdot \mathbf{x}, \mathbf{a} \cdot \mathbf{y})) > 1\}$,
- $\mathcal{R} = \{(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \mid M > P(\gcd(\mathbf{a} \cdot \mathbf{x}, \mathbf{a} \cdot \mathbf{y})) > Q\}$,
- $\mathcal{S} = \{(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \mid P(\gcd(\mathbf{a} \cdot \mathbf{x}, \mathbf{a} \cdot \mathbf{y})) \geq M\}$,
- $\mathcal{T} = \{(\mathbf{x}, \mathbf{y}) \in \mathcal{L} \mid p \mid \gcd(\mathbf{a} \cdot \mathbf{x}, \mathbf{a} \cdot \mathbf{y}) \text{ for some } p \leq Q\}$.

Obviously $\mathcal{Q} \subseteq \mathcal{T}$, and

$$1 - \rho_{\mathbf{a}}(M) = M^{-2m} (\#\mathcal{Q} + \#\mathcal{R} + \#\mathcal{S}) \leq M^{-2m} (\#\mathcal{T} + \#\mathcal{R} + \#\mathcal{S}).$$

For an integer $d \geq 1$, let us denote by $\mathcal{U}_d(M)$ the set of all integer vectors $\mathbf{x} \in \mathbb{N}^m$ with $h(\mathbf{x}) \leq M$ and $d \mid \mathbf{a} \cdot \mathbf{x}$, and put $U_d(M) = \#\mathcal{U}_d(M)$. Because $\gcd(a_1, \dots, a_m) = 1$, we obviously have $U_p(p) = p^{m-1}$ for any prime p . Then, for any squarefree d , by the Chinese Remainder Theorem, we conclude that $U_d(d) = d^{m-1}$, and $U_d(dK) = K^m d^{m-1}$ for any integer K . Finally, using $U_d(d \lfloor M/d \rfloor) \leq U_d(M) \leq U_d(d \lceil M/d \rceil)$, we obtain that for $d = o(M/m)$,

$$\begin{aligned} U_d(M) &= (M/d + O(1))^m d^{m-1} = \frac{M^m}{d} (1 + O(d/M))^m \\ &= \frac{M^m}{d} \exp(O(dm/M)) = \frac{M^m}{d} (1 + O(md/M)). \end{aligned} \tag{3}$$

It is also clear that for any prime p

$$U_p(M) \leq (M/p + 1)M^{m-1} = M^m/p + M^{m-1}. \tag{4}$$

By the inclusion exclusion principle we have

$$M^{2m} - \#\mathcal{T} = \sum_{\substack{d \geq 1 \\ 1 \leq P(d) \leq Q}} \mu(d) U_d(M)^2$$

where μ is the Möbius function. We recall that $\mu(1) = 1$, $\mu(d) = 0$ if $d \geq 2$ is not squarefree, and $\mu(d) = (-1)^{\nu(d)}$ otherwise, where $\nu(d)$ is the number of prime divisors of d ; see Section 2.1 of [1]. From the definition of \mathcal{Q} we see that any squarefree d with $P(d) \leq Q$ does not exceed $(M/m)^{1/2}$. Now from (3) we derive that for such d ,

$$U_d(M)^2 = \frac{M^{2m}}{d^2} (1 + O(md/M)) = \frac{M^{2m}}{d^2} + O(mM^{2m-1}/d).$$

Therefore

$$\begin{aligned}
 M^{2m} - \#\mathcal{T}(M) &= \sum_{\substack{d>1 \\ 1 \leq P(d) \leq Q}} \mu(d) \left(\frac{M^{2m}}{d^2} + O(mM^{2m-1}/d) \right) \\
 &= M^{2m} \sum_{\substack{d>1 \\ 1 \leq P(d) \leq Q}} \frac{\mu(d)}{d^2} + O\left(mM^{2m-1} \sum_{d \leq (M/m)^{1/2}} d^{-1} \right) \\
 &= M^{2m} \prod_{p \leq Q} \left(1 - \frac{1}{p^2} \right) + O(mM^{2m-1} \ln(M/m)).
 \end{aligned}$$

We now recall that

$$\prod_{p \leq Q} \left(1 - \frac{1}{p^2} \right) = \prod_p \left(1 - \frac{1}{p^2} \right) + O(Q^{-1}) = \zeta(2)^{-1} + O(Q^{-1})$$

see Section 11.4 of [1]. Thus

$$\#\mathcal{T} = (1 - \zeta(2)^{-1})M^{2m} + O(M^{2m}Q^{-1} + mM^{2m-1} \ln(M/m)).$$

When $2M/\ln^2 M \geq m$, then the last term is smaller than the last but one term.

Thus

$$\#\mathcal{T} = (1 - \zeta(2)^{-1})M^{2m} + O(M^{2m}Q^{-1}).$$

For $\#\mathcal{R}$, using (4), and the inequality $(a + b)^2 \leq 2(a^2 + b^2)$ we get

$$\begin{aligned}
 \#\mathcal{R} &\leq \sum_{Q < p < M} U_p(M)^2 \leq 2 \sum_{Q < p < M} \left(\frac{M^{2m}}{p^2} + M^{2m-2} \right) \\
 &\leq 2M^{2m} \sum_{k > Q} \frac{1}{k^2} + 2M^{2m-2} \sum_{k < M} 1 \\
 &= O(M^{2m}Q^{-1} + M^{2m-1}) = O(M^{2m}Q^{-1}).
 \end{aligned}$$

Finally, using (4) again, we derive

$$\begin{aligned}
 \#\mathcal{S} &\leq \sum_{p \geq M} U_p(M)^2 \leq M^{m-1} \sum_{p \geq M} U_p(M) \\
 &= M^{m-1} \sum_{h(\mathbf{x}) \leq M} \sum_{\substack{p \geq M \\ p | \mathbf{a} \cdot \mathbf{x}}} 1 = M^{m-1} \sum_{h(\mathbf{x}) \leq M} \nu(\mathbf{a} \cdot \mathbf{x}) \\
 &= O\left(M^{m-1} \sum_{h(\mathbf{x}) \leq M} \ln \mathbf{a} \cdot \mathbf{x} \right),
 \end{aligned}$$

because for any integer $k \geq 2$ we have $\nu(k) = O(\ln k / \ln \ln k)$. Taking into account that $\mathbf{a} \cdot \mathbf{x} \leq mMN$ we finish the proof. \square

Corollary 2 . . . $\mathbf{a} \in \mathbb{Z}^m$ N M
 $M / \max\{m, \ln N\} \rightarrow \infty$

$$\rho_{\mathbf{a}}(M) \geq \zeta(2)^{-1} + o(1).$$

3 Algorithmic Implications

It is easy to see that Corollary 2 implies that for any a_1, \dots, a_m one can compute $\gcd(a_1, \dots, a_m)$ probabilistically as the gcd of two integers of asymptotically the same bit lengths as the original data, while the result of [2] only guarantees the same for two integers of bit lengths twice more. The probability of success in both cases is, asymptotically, at least $\zeta(2)^{-1} = 6/\pi^2 = 0.6079\dots$. Repeating this several times and choosing the smallest result one gets an efficient and reliable algorithm to compute the above gcd which is an attractive alternative to the m -step (deterministic) chain of computation

$$\begin{aligned} \gcd(a_1, \dots, a_m) &= \gcd(\gcd(a_1, a_2), a_3, \dots, a_m) \\ &= \gcd(\dots (\gcd(\gcd(a_1, a_2), a_3), \dots, a_m)). \end{aligned}$$

For illustration, we take l -bit primes p_1, \dots, p_m , $a = p_1 \cdots p_m$, and $a_i = a/p_i$ for $i \leq m$. Then indeed $m - 1$ steps are necessary until the gcd, which equals 1, is found.

After $i - 1$ steps, the current value of the gcd has about $(m - i)l$ bits, and the reduction of the $(m - 1)l$ -bit a_{i+1} modulo this gcd takes about $2l^2(m - i)(i - 1)$ operations in naive arithmetic; see [3], Section 2.4. This comes to a total of about $l^2m^3/3$ operations. If one gcd of n -bit integers costs about cn^2 operations, for a constant c , then all the gcds required amount to $cm^3/3$, for a grand total of $l^2m^3(1 + c)/3$ operations.

In our algorithm, we can choose x_i and y_i of $\ln(ml)$ bits. The inner products together cost just over $2lm^2 \ln(ml)$ operations, and the single gcd about cl^2m^2 . The latter is the dominant cost, and thus our algorithm is faster by a factor of about $m/3$ than the standard one.

In other words, if $k < m/3$, maybe $k \approx \sqrt{m}$, and confidence at least $1 - \zeta(2)^{-k}$ is sufficient, then the k -fold repetition of our algorithm is faster. (In practice, one would not just repeat, but reduce the inputs modulo the gcd candidate obtained so far, and either find that it divides all of them and thus is the true gcd, or continue with the smaller values.)

The advantage of our method evaporates when one uses fast arithmetic.

The worst-case example is not quite as esoteric as it may look. In resultant and subresultant computations with several integer polynomials in several variables, nontrivial gcds occur with definite patterns.

4 Conclusion and Open Questions

It would be interesting to evaluate the constant implicit in the bound of Theorem 1. This should be possible, but may involve some nontrivial amount of technical details.

We believe that in fact $\rho_{\mathbf{a}}(M) \sim \zeta(2)^{-1}$ under the condition of Corollary 2 (or some similar conditions maybe marginally more restrictive). We believe that better sieving technique should produce such a result. Although it may have no algorithmic application it is a natural question which would be interesting to resolve.

Finally, we remark that the approach of [2] leads to an algorithm for an extended gcd problem; see [3] for the background on this problem. Namely, solving the the extended gcd problem for $\mathbf{a} \cdot \mathbf{x}$ and $\mathbf{a} \cdot \mathbf{x}$ we obtain a relation

$$c_1 a_1 + \dots + c_n a_n = d$$

for some integers c_1, \dots, c_n, d with $d > 0$. Repeating this the appropriate number of times, given by (2), and choosing the relation with the smallest value of d , we solve the extended gcd problem with probability at least $1 - \delta$.

References

1. T. M. Apostol, *Introduction to analytic number theory*, Springer-Verlag, NY, 1976.
2. G. Cooperman, S. Feisel, J. von zur Gathen and G. Havas, ‘GCD of many integers’, *Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **1627** (1999), 310–317.
3. J. von zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge University Press, Cambridge, 2003.
4. G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, Oxford Univ. Press, Oxford, 1979.

On the Hardness and Easiness of Random 4-SAT Formulas

Andreas Goerdt and André Lanka

Technische Universität Chemnitz, Fakultät für Informatik,
Straße der Nationen 62, 09107 Chemnitz, Germany
{goerdt, lanka}@informatik.tu-chemnitz.de

Abstract. Assuming random 3-SAT formulas are hard to refute, Feige showed approximation hardness results, among others for the max bipartite clique. We extend this result in that we show that approximating max bipartite clique is hard under the weaker assumption, that random 4-SAT formulas are hard to refute. On the positive side we present an efficient algorithm which finds a hidden solution in an otherwise random not-all-equal 4-SAT instance. This extends analogous results on not-all-equal 3-SAT and classical 3-SAT. The common principle underlying our results is to obtain efficiently information about discrepancy (expansion) properties of graphs naturally associated to 4-SAT instances. In case of 4-SAT (or k -SAT in general) the relationship between the structure of these graphs and that of the instance itself is weaker than in case of 3-SAT. This causes problems whose solution is the technical core of this paper.

1 Introduction and Results

1.1 Some Terminology

Given a standard set of n propositional variables $\text{Var} = \text{Var}_n$ a k -clause is an ordered k -tuple $l_1 \vee \dots \vee l_k$ where $l_i = x$ or $l_i = \neg x$ for an $x \in \text{Var}_n$. We denote the variable underlying the literal l by $\text{Var}(l)$. Lit_n is the set of literals over Var_n . Altogether we have $2^k n^k$ different k -clauses. A k -SAT formula F simply is a set of k -clauses. Given a truth value assignment a , a satisfies F if each clause of F becomes true under a . A clause C is true in the not-all-equal sense under the truth value assignment a if it contains one literal which evaluates to true and another one which evaluates to false under a . The problem to decide satisfiability in the not-all-equal sense for 3-SAT formulas is \mathcal{NP} -complete.

Considering any high probability event, that is whose probability goes to 1 when n goes to infinity, where we have an underlying family of probability spaces, one for each n , the following certification problem naturally arises: Given a random instance, how can we be sure that this event really holds for the instance at hand? This question can usually be answered running appropriate (inefficient) algorithms with the given instance. We however are interested in an efficient algorithm satisfying the following requirements: It always stops in polynomial time.

It says that the instance belongs to the event considered or it gives an inconclusive answer. If the answer is not the inconclusive one the answer must be correct, that is we have a certificate for the event. Moreover the algorithm must be complete, in that it gives the correct answer with high probability with respect to the random instance. In this case we speak of “efficient certification”.

1.2 The Hardness Result

Given $p = p(n)$ with $0 \leq p \leq 1$ the random formula $\text{Form}_{n,k,p}$ is obtained as follows: Pick each of the $2^k n^k$ k -clauses independently with probability p . $\text{Form}_{n,k,c/n^{k-1}}$ is unsatisfiable with high probability when $c > \ln 2$ is a constant. This follows from a simple first moment calculation of the number of satisfying assignments. Thus almost all (that is with high probability) formulas are unsatisfiable, but we have no efficient algorithm to certify this. Feige [6] introduces the random 3-SAT hardness hypothesis: For any constant $c > \ln 2$ there is no efficient certification algorithm of the unsatisfiability of $\text{Form}_{n,3,c/n^2}$. The truth of this hypothesis is supported by the fact that for $p(n) = o(1/n^{3/2})$ no progress concerning the efficient certification of unsatisfiability of $\text{Form}_{n,3,p}$ has been made. The best result known is efficient certification of unsatisfiability of $\text{Form}_{n,3,c/n^{3/2}}$ for some sufficiently large constant c , cf. [7]. Feige shows that the random 3-SAT hardness hypothesis implies several lower bounds on the approximability of combinatorial problems for which such bounds could not be obtained from worst-case assumptions like $\mathcal{P} \neq \mathcal{NP}$. As a random hardness hypothesis is much stronger than a mere worst-case hypothesis like $\mathcal{P} \neq \mathcal{NP}$ it is particularly important to weaken it as much as possible. This motivates to consider random 4-SAT instead of 3-SAT. The random 4-SAT hardness hypothesis reads: For any constant $c > \ln 2$ there is no efficient certification algorithm of the unsatisfiability for $\text{Form}_{n,4,c/n^3}$. The trivial reduction: Given $F = \text{Form}_{n,3,c/n^2}$ place a random literal into each clause of F to obtain a 4-SAT instance G , shows, that the 3-SAT hypothesis is stronger than the 4-SAT hypothesis.

Among the problems considered by Feige is the max clique problem for bipartite graphs. Let $G = (V_1, V_2, E)$ be a bipartite graph. V_1 and V_2 are the sets of vertices (V_1 is the left hand side and V_2 is the right hand side) and $E \subseteq V_1 \times V_2$ is the set of edges. A (bipartite) clique in G is a subgraph $H = (W_1, W_2, F)$ of G with $W_i \subseteq V_i$ such that $F = W_1 \times W_2$. Sometimes we denote such a clique simply by (W_1, W_2) . We are interested in the optimization problem maximum clique, that is to determine the maximum size of a clique in G . When the size of H is measured as # of vertices of $H = |W_1| + |W_2|$ the problem is solvable in polynomial time [9], problem GT24. If however the size of H is measured as # of edges of $H = |W_1 \times W_2| = |W_1| \cdot |W_2|$ the problem interestingly becomes \mathcal{NP} -hard [16] and approximation algorithms are of interest. This is the version of the problem we consider. The approximation ratio of an algorithm for a maximization problem is the maximum size of a solution divided by the size of the solution found by the algorithm. For the classical clique problem no approximation ratio below $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ is possible by a polynomial time algorithm (unless $\mathcal{P} = \mathcal{NP}$), cf. [13]. Interesting enough, such results are not

known for the bipartite case. Feige shows in [6] that there is a constant $\delta > 0$ such that the bipartite clique problem cannot be approximated with a ratio below n^δ , provided the random 3-SAT hardness hypothesis holds. Our hardness result is

Theorem 1. *Let $\delta > 0$ be a constant. For every $n \geq n_0(\delta)$ there exists a constant $c = c(\delta) > 0$ such that for every $\varepsilon > 0$ there is a constant $n_0(\delta, \varepsilon) > n_0(\delta)$ such that for every $n \geq n_0(\delta, \varepsilon)$ and every graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2| = n$ and $|E| \geq (n/16)^2(1 + \varepsilon_1)$ and $|E| \leq (n/16)^2(1 + \varepsilon_2)$ there exists a set S of size $|S| \geq cn^\delta$ such that S is a clique in G .*

The technical heart of the proof of Theorem 1 is the subsequent Theorem 2 from which Theorem 1 is obtained by means of the derandomized graph product [1]. For the proof see [12].

Theorem 2. *Let $\varepsilon_1 > \varepsilon_2 > 0$ be constants. For every $n \geq n_0(\varepsilon_1, \varepsilon_2)$ and every graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2| = n$ and $|E| \geq (n/16)^2(1 + \varepsilon_1)$ and $|E| \leq (n/16)^2(1 + \varepsilon_2)$ there exists a set S of size $|S| \geq cn^\delta$ such that S is a clique in G .*

1.3 The Easiness Result

Given an assignment ϕ we let $CT_i = CT_{i,\phi}$ be the set of all clauses with exactly i literals true ($= 1$) under ϕ and $4 - i$ false ($= 0$). We have that $|CT_i| = \binom{4}{i}n^4$. We let $CT_{nae} = CT_{nae,\phi} = CT_1 \cup CT_2 \cup CT_3$ be the set of all clauses satisfied by ϕ in the not-all-equal sense. We describe the way to generate our random formula I . To this end let $0 < \eta_1, \eta_2, \eta_3 < 1$ with $\eta_1 + \eta_2 + \eta_3 = 1$ be three constants and let $d = d_{\eta_1, \eta_2, \eta_3}$ be a (large) constant. We let $p_i = \eta_i d / n^3$ be three probabilities.

1. We pick any assignment ϕ of Var_n . (This is the hidden solution.) Note that ϕ need not be a random assignment, just any assignment. Let $M = CT_{nae,\phi}$.
2. Pick a uniform random clause $C \in M$ and delete it from M . Include C in the random instance I with probability p_i iff $C \in CT_{i,\phi}$.
3. Repeat 2. until $M = \emptyset$.

All instances generated are satisfiable in the not-all-equal sense and we are left with a classical certification problem. Such certification problems have a long tradition. Seminal work has been done by [2] in that spectral techniques have been introduced to find a hidden 3-coloring in a sparse random graph, that is with a linear number of edges. Note that usually hidden solutions in denser instances are easier to find because the structure gives more information. This approach has been further developed to 2-colorings of random 3-uniform hypergraphs (or not all equal 3-SAT instances) with a linear number of edges (clauses) [3] and – recently – to hidden satisfying assignments in a random 3-SAT formula [8]. By developing this approach further we show

Theorem 3. *Let $0 < \eta_i < 1$ be constants and let $d = d_{\eta_1, \eta_2, \eta_3}$ be a constant. For every $n \geq n_0(\eta_1, \eta_2, \eta_3, d)$ and every graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2| = n$ and $|E| \geq (n/16)^2(1 + \varepsilon_1)$ and $|E| \leq (n/16)^2(1 + \varepsilon_2)$ there exists a set S of size $|S| \geq cn^\delta$ such that S is a clique in G .*

2 Proof of Theorem 2

2.1 Discrepancy Certification in Random Bipartite Graphs

Let $B = (V_1, V_2, E)$ be a bipartite graph on $2n$ vertices with $|V_1| = |V_2| = n$. Let $E(X, Y) = \{\{x, y\} \in E \mid x \in V_1, y \in V_2\}$ be the set of edges with one endpoint in $X \subseteq V_1$ and the other one in $Y \subseteq V_2$. We abbreviate $|E(X, Y)|$ with $e(X, Y)$. We say B as above is of discrepancy ε iff for all $X \subseteq V_1, |X| = \alpha n$ and all subsets $Y \subseteq V_2, |Y| = \beta n$ we have that $|e(X, Y) - \alpha\beta \cdot |E|| \leq \varepsilon|E|$.

The random bipartite graph $B_{n,c/n}$ has the set of vertices $V_1 = \{1, \dots, n\}$ and $V_2 = \{n + 1, \dots, 2n\}$. Each edge $\{x, y\}$ with $x \in V_1$ and $y \in V_2$ is picked with probability c/n independently. Item (a) of the following fact follows by a simple Chernoff bound consideration, (b) is from [4].

Fact 4. (a) For any $\varepsilon > 0$ there exists $c = c(\varepsilon)$ such that $B_{n,c/n}$ is of discrepancy ε with probability $1 - o(1)$.
 (b) For any $\varepsilon > 0$ there exists $c = c(\varepsilon)$ such that $B_{n,c/n}$ is of discrepancy ε with probability $1 - o(1)$.

2.2 Proof of Theorem 2

Now let $P = P(F)$ be the set of positive clauses of F , i.e. clauses in F containing only non-negated variables. The clauses containing only negated variables form the set $N = N(F)$ and are called negative clauses. For the truth value assignment a let T_a be the set of variables set to true by a and F_a be the set of variables false under a . In order that a satisfies F we must have that $T_a \cap C \neq \emptyset$ for all $C \in P(F)$ and $F_a \cap C \neq \emptyset$ for all $C \in N(F)$.

A simple Chernoff bound argument shows that $T_a \geq (1 - \varepsilon)n$ and $F_a \geq (1 - \varepsilon)n$ in order that the preceding properties hold for $F = \text{Form}_{n,4,c/n^3}$ with high probability if $c = c(\varepsilon)$ is large enough. But of course we cannot certify this, only a weaker property. Consider the following algorithm: Pick any clause $C \in P$, delete all clauses $D \in P$ with $C \cap D \neq \emptyset$ from P , and continue in this way until $P = \emptyset$.

Now let S be the set of variables belonging to the clauses picked, then $S \cap C \neq \emptyset$ for any $C \in P$. Thus with high probability $|S| \geq (1 - \varepsilon)n$. Each set $T \subseteq S$ such that $T \cap D \neq \emptyset$ for all clauses D picked must satisfy $|T| \geq |S|/4$. Proceeding in the same way for $N = N(F)$ we can efficiently certify the following property: If a satisfies $F = \text{Form}_{n,4,c/n^3}$ then $|T_a|, |F_a| \geq n/5$.

The proof of Theorem 2 relies on the certification of low discrepancy of certain bipartite projection graphs of $\text{Form}_{n,4,c/n^3}$. Let F be a 4-SAT formula and $S \subseteq F$ an arbitrary set of clauses from F . Then we define 6 projection graphs $B_{ij} = (V_1, V_2, E_{ij}), 1 \leq i < j \leq 4$, of S . The sets V_1 and V_2 are copies of the variables Var of F . We have the edge $\{x, y\} \in E_{ij}$ with $x \in V_1$ and $y \in V_2$ if and only if we have a clause $l_1 \vee l_2 \vee l_3 \vee l_4 \in S$ with $\text{Var}(l_i) = x$ and $\text{Var}(l_j) = y$.

Algorithm 5.

Input: A 4-SAT formula F and $\varepsilon > 0$.

1. Check that $|T_a| \geq n/5$ and $|F_a| \geq n/5$ for any satisfying assignment a of F . Give an inconclusive answer if one inequality cannot be certified as above.
2. Check that $|P| = cn \cdot (1 + o(1))$ and $|N| = cn \cdot (1 + o(1))$. Construct the 6 projection graphs of N and the 6 projection graphs of P . Check for every projection that the number of edges is $\geq |N| \cdot (1 - o(1))$ for N and $\geq |P| \cdot (1 - o(1))$ for P . Give an inconclusive answer if this is not the case.
3. Apply the Algorithm **BipDisc** from Section 2 to certify low discrepancy with respect to $\varepsilon > 0$ for all these projection graphs. Give an inconclusive answer if one application gives an inconclusive answer. Give a positive answer otherwise.

A simple estimate using Chernoff’s bound and Markov’s inequality shows that Step 2 is complete for $\text{Form}_{n,4,c/n^3}$. So Algorithm 5 gives almost surely a positive answer for $\text{Form}_{n,4,c/n^3}$.

For $X_i \subseteq \text{Var}$ we say that a clause $C = l_1 \vee l_2 \vee l_3 \vee l_4$ is of $\text{type}(C) = (X_1, X_2, X_3, X_4)$ iff $\text{Var}(l_i) \in X_i$ for all i . For a set of clauses S we let $(X_1, X_2, X_3, X_4)_S = \{C \in S \mid \text{type}(C) = (X_1, X_2, X_3, X_4)\}$.

We let $\varrho = |P| = |P(F)|$ and $\nu = |N| = |N(F)|$. Then $\varrho_i = \varrho_{i,a}$ is the number of clauses of P which contain exactly i literals true under a . We use the analogous notation $\nu_i = \nu_{i,a}$ for N .

Theorem 6. $\varepsilon > 0$

Let F be a 4-SAT formula in $\text{Form}_{n,4,c/n^3}$ and let a satisfy F with $|F_a| = \alpha n$.

(a)

$$\begin{aligned} \varrho_0 &= 0 & \varrho_3 &= (-12\alpha^2 + 4\alpha)\varrho + 3\varrho_1 + O(\varepsilon)\varrho \\ \varrho_2 &= 6\alpha^2\varrho - 3\varrho_1 + O(\varepsilon)\varrho & \varrho_4 &= (6\alpha^2 - 4\alpha + 1)\varrho - \varrho_1 + O(\varepsilon)\varrho \end{aligned}$$

(b)

$$\begin{aligned} \nu_i &= \alpha \nu & \nu_i &= (1 - \alpha) \nu \\ \nu_0 &= 0 & \nu_3 &= (-12(1 - \alpha)^2 + 4(1 - \alpha))\nu + 3\nu_1 + O(\varepsilon)\nu \\ \nu_2 &= 6(1 - \alpha)^2\nu - 3\nu_1 + O(\varepsilon)\nu & \nu_4 &= (6(1 - \alpha)^2 - 4(1 - \alpha) + 1)\nu - \nu_1 + O(\varepsilon)\nu \end{aligned}$$

(c) $1/3 - O(\varepsilon) \leq \alpha \leq 2/3 + O(\varepsilon)$

Note that (a) implies that ϱ_2, ϱ_3 and ϱ_4 are determined by α and ϱ_1 up to the $O(\varepsilon)$ -terms. The same applies to the ν_i . Note, the claim of Theorem 6 is only useful if α is substantial larger than ε . This shows the relevance of Step 1 in Algorithm 5. It certifies that α is bounded away from 0 by a fixed constant. This fact allows us to find a sufficiently small constant $\varepsilon > 0$.

To show that Algorithm 5 correctly certifies the properties of Theorem 6 let $\varepsilon > 0$ be a constant and F be a 4-SAT formula which passes the algorithm successfully. Let a satisfy F with $|F_a| = \alpha n$. The first equation $\varrho_0 = 0$ trivially holds.

By low discrepancy we get for any projection B of $P = P(F)$ that $e_B(F_a, F_a) = \alpha^2 \cdot \varrho + O(\varepsilon)\varrho$. No clause from ϱ_3 induces an edge belonging to $E_B(F_a, F_a)$. Looking at all 6 projections each clause from ϱ_2 induces one edge in one projection and each clause from ϱ_1 induces one edge in three projections. Thus we have

$$6\alpha^2\varrho + O(\varepsilon) \cdot \varrho = \sum_B e_B(F_a, F_a) = 3\varrho_1 + 1\varrho_2 + o(\varrho), \tag{1}$$

where B ranges over all 6 projection of P . The $o(\varrho)$ term accounting for those pairs of clauses inducing the same edge. In each projection B of P we have $e_B(T_a, T_a) = (1 - \alpha)^2 \cdot \varrho + O(\varepsilon)\varrho$ and therefore

$$6(1 - \alpha)^2 \cdot \varrho = 6\varrho_4 + 3\varrho_3 + \varrho_2 + O(\varepsilon)\varrho. \tag{2}$$

Finally

$$\varrho = \varrho_4 + \varrho_3 + \varrho_2 + \varrho_1. \tag{3}$$

Remember, $\varrho_0 = 0$ as a is a satisfying assignment. The equations from (a) now follow by the equations (1), (2) and (3).

(b) follows analogously with N and $|T_a| = (1 - \alpha)n$.

(c) The upper bound can be obtained by $\varrho \geq \varrho_1 + \varrho_4$ and the lower bound by $\nu \geq \nu_1 + \nu_4$.

□

To extend the construction from Section 4.1 of [6] from 3-SAT to 4-SAT is the purpose of

Definition 7.

Let V_1, V_2 be two disjoint sets of variables. Let E be the set of clauses $\{C, D\} \in E \mid C = u_1 \vee u_2 \vee u_3 \vee u_4, D = v_1 \vee v_2 \vee v_3 \vee v_4, \text{Var}(u_i) \neq \text{Var}(v_i)$

As we consider clauses as ordered it can be well that $\{x_1 \vee x_2 \vee x_3 \vee x_4, \neg x_2 \vee \neg x_1 \vee x_4 \vee x_3\} \in E$ provided the x_i are all distinct. However we never have that $\{x_1 \vee x_2 \vee x_3 \vee x_4, \neg x_1 \vee v_1 \vee v_2 \vee v_3\} \in E$ as $\text{Var}(x_1) = \text{Var}(\neg x_1) = x_1$.

For a set of clauses S and $1 \leq i \leq 4$ the rotations of S are:

$$\begin{aligned} \text{ROT}_1(S) &= \{v_2 \vee v_3 \vee v_4 \vee v_1 \mid v_1 \vee v_2 \vee v_3 \vee v_4 \in S\} \\ \text{ROT}_2(S) &= \{v_3 \vee v_4 \vee v_1 \vee v_2 \mid v_1 \vee v_2 \vee v_3 \vee v_4 \in S\} \\ \text{ROT}_3(S) &= \{v_4 \vee v_1 \vee v_2 \vee v_3 \mid v_1 \vee v_2 \vee v_3 \vee v_4 \in S\} \\ \text{ROT}_4(S) &= S \end{aligned}$$

Corollary 8.

Let $\delta > 0$ ($\delta = 1/50$) and $c > 0$. Let $F = \text{Form}_{n,4,c/n^3}$. Then $\nu(F) \geq (cn/16)^2 \cdot (1 + \delta)$

$BG(P, \text{ROT}_i(N)), BG(P, \text{ROT}_j(P)), BG(N, \text{ROT}_j(N)),$ with $i = 1 \dots 4$ and $j = 1, 2$.

We only need to show that Algorithm 5 correctly certifies the property claimed. To this end let F be a 4-SAT formula which passes Algorithm 5 successfully. We distinguish two cases. In the first case is $\varrho_2 \leq 3/8 \cdot \varrho \cdot (1 + \delta)$ and $\nu_2 \leq 3/8 \cdot \nu \cdot (1 + \delta)$. In the second case at least one inequality is violated. We start with the second case.

Assume $\varrho_2 > 3/8 \cdot \varrho(1 + \delta)$. Note that ϱ_2 refers to six subsets of clauses containing two variables true and two variables false under a . So there is at least one subset with cardinality $\geq 1/16 \cdot \varrho(1 + \delta)$. Let for example $(T_a, F_a, F_a, T_a)_P$ be this subset. Then $BG(P, ROT_2(P))$ has a large bipartite clique. For the left side of the clique take all clauses of type $(T_a, F_a, F_a, T_a)_P$ in P . The right side is the rotated set of these clauses. Through the rotation the clauses change to $(F_a, T_a, T_a, F_a)_{ROT_2(P)}$. As $T_a \cap F_a = \emptyset$, $(T_a, F_a, F_a, T_a)_P$ and $(F_a, T_a, T_a, F_a)_{ROT_2(P)}$ form a bipartite clique. The size of the clique is bounded below by

$$(1/16 \cdot \varrho(1 + \delta))^2 \geq (cn/16)^2 \cdot (1 + \delta)^2 \cdot (1 - o(1)) > (cn/16)^2 \cdot (1 + \delta).$$

For any of the five other types we get the same bound maybe using $BG(P, ROT_1(P))$. If $\nu_2 > 3/8 \cdot \nu(1 + \delta)$ use $BG(N, ROT_1(N))$ resp. $BG(N, ROT_2(N))$ in the same way.

Now we come to the case $\varrho_2 \leq 3/8 \cdot \varrho(1 + \delta)$ and $\nu_2 \leq 3/8 \cdot \nu(1 + \delta)$ From the equations for ϱ_2 and ν_2 in Theorem 6 we get

$$\varrho_1 = 2\alpha^2 \varrho - \varrho_2/3 + O(\varepsilon)\varrho \geq 2\alpha^2 \varrho - 1/8 \cdot \varrho(1 + \delta) + O(\varepsilon)\varrho$$

and

$$\nu_1 \geq 2(1 - \alpha)^2 \nu - 1/8 \cdot \nu(1 + \delta) + O(\varepsilon)\nu.$$

As ϱ_1 consists of four subsets of clauses having exactly one variable true under a we have one subset with cardinality $\geq (\alpha^2/2 - (1 + \delta)/32 + O(\varepsilon))\varrho$. For example this is $(F_a, T_a, F_a, F_a)_P$. Also we get one subset in N having exactly one variable false under a with at least $((1 - \alpha)^2/2 - (1 + \delta)/32 + O(\varepsilon))\nu$ clauses. Let this subset be $(F_a, T_a, T_a, T_a)_N$. Looking at $BG(P, ROT_3(N))$ we see that these two subsets form a bipartite clique with at least

$$\left(\frac{\alpha^2}{2} - \frac{1 + \delta}{32} + O(\varepsilon)\right) \varrho \cdot \left(\frac{(1 - \alpha)^2}{2} - \frac{1 + \delta}{32} + O(\varepsilon)\right) \nu \tag{4}$$

edges. Conceive (4) as a function of α . Then it is concave for $1/5 \leq \alpha \leq 4/5$. Theorem 6 gives us $1/3 - O(\varepsilon) \leq \alpha \leq 2/3 + O(\varepsilon)$ as a is a satisfying assignment. Because of the concavity we only have to check these both limits to lower bound (4). For ε and δ sufficiently small we are able to lower bound both cases by $(cn)^2 \cdot (1 + o(1))/250 > (cn/16)^2 \cdot (1 + \delta)$. \square

Theorem 9. $\dots \varepsilon > 0 \dots \dots \dots c = c(\varepsilon) \dots$
 $\dots \dots F = \text{Form}_{n,4,c/n^3} \dots \dots \dots (cn/16)^2 \cdot (1 + \varepsilon) \dots$
 $\dots \dots BG(R, T) \dots \dots R \dots \dots T \dots \dots \dots i(N(F)),$
 $\dots \dots i(P(F)) \dots \dots 1 \leq i \leq 4 (R = T \dots \dots)$

The proof follows by a Chernoff bound argument and can be found in [12]. Corollary 8 and Theorem 9 shows the correctness of Theorem 2. If we would have an approximation algorithm with ratio for example 1.01, we could distinguish between the satisfiable formulas inducing graphs with cliques $\geq (cn/16)^2 \cdot (1.02)$ (Corollary 8) and the typical formulas whose graphs only have cliques of size e.g. $(cn/16)^2 \cdot (1.001)$ from Theorem 9. This means we could refute random 4-SAT.

3 Proof of Theorem 3

3.1 The Algorithm

For $U_i \subseteq \text{Var}$ we say that a clause $l_1 \vee l_2 \vee l_3 \vee l_4$ is of $\dots \{U_1, U_2, U_3, U_4\}$ if there is a permutation $g_1 \vee g_2 \vee g_3 \vee g_4$ of the l_i such that $\text{Var}(g_i) \in U_i$. Remember $\text{Var}(g_i)$ denotes the variable underlying the literal g_i . Note, the definition of the type is slightly different to that above Theorem 6. Given a 4-SAT formula F , $\{U_1, U_2, U_3, U_4\}_F$ is the set of clauses of type $\{U_1, U_2, U_3, U_4\}$ in F . We write $\{U_1, U_2, -, -\} = \{U_1, U_2, \text{Var}, \text{Var}\}$ and $\{U_1, U_2, -, -\}_F$ then stands for the subset of clauses C of F in which we have two positions one of which is filled with a literal over U_1 and the other one with a literal over U_2 . (Note that the literals in the two positions can be equal.)

Given a formula F and an assignment ϕ we let $CT_i(F) = CT_{i,\phi}(F) = F \cap CT_{i,\phi}$ be the set of those clauses of F with exactly i literals true under ϕ . The support in CT_1 of the variable x with respect to F and ϕ is

$$\text{Supp}_{1,F,\phi}(x) = |\{C \in CT_{1,\phi}(F) \mid x \in C \text{ and } \phi(x) = 1 \text{ or } \neg x \in C, \phi(x) = 0\}|.$$

Similarly for CT_3 we have

$$\text{Supp}_{3,F,\phi}(x) = |\{C \in CT_{3,\phi}(F) \mid x \in C \text{ and } \phi(x) = 0 \text{ or } \neg x \in C, \phi(x) = 1\}|.$$

Thus $\text{Supp}_{F,\phi}(x) = \text{Supp}_{1,F,\phi}(x) + \text{Supp}_{3,F,\phi}(x)$ is the number of clauses of F which have exactly one literal true or exactly one literal false under ϕ and this literal is either x or $\neg x$. $\text{Occ}_F(x) = \text{Occ}_F(\neg x) = |\{C \in F \mid x \in C \text{ or } \neg x \in C\}|$ is the number of clauses of F which contain x or $\neg x$. Note that $\text{Occ}(x)$ need not be equal to the number of actual occurrences of $x, \neg x$ as $x, \neg x$ can occur several times inside a clause.

Recall the generation procedure of our formulas from Subsection 1.3. Let ϕ be the assignment picked in Step 1, then we have that $|CT_{i,\phi}(I)|$ follows the binomial distribution with parameters $|CT_{i,\phi}|$ and p_i . For the expectation we have $E[|CT_i(I)|] = p_i \cdot 4n^4$ for $i = 1, 3$ and $E[|CT_2(I)|] = p_2 \cdot 6n^4$. For $x \in \text{Var}$ we can decompose $\text{Occ}_I(x) = X_1 + X_2 + X_3$ where X_i follows the binomial distribution with parameters $4n^4 - 4(n-1)^4 = 16n^3 + O(n^2)$ and p_i for $i = 1, 3$. X_2 follows the binomial distribution with $6n^4 - 6(n-1)^4 = 24n^3 + O(n^2)$. We have that

$$E[\text{Occ}_I(x)] = (16\eta_1 + 24\eta_2 + 16\eta_3)d + O(1/n).$$

We let $\mu = 16\eta_1 + 24\eta_2 + 16\eta_3$ and $d_i = \eta_i d$ throughout. $\text{Supp}_{F,\phi}(x) = Y_1 + Y_3$ where Y_i follows the binomial distribution with parameter $4n^3 + O(n^2)$ and p_i .

Then $E[\text{Supp}(x)] = 4\eta d + O(1/n)$ where $\eta = \eta_1 + \eta_3$ throughout. $R(I)$ is the subset of those variables for which $\text{Occ}_I(x)$ and $\text{Supp}_{I,\phi}(x)$ are approximately right (like the expectation). Given an assignment ϕ and $\varepsilon > 0$ we define

$$R(I) = R_{\phi,\varepsilon}(I) = \{x \in V \mid |\text{Occ}_I(x) - \mu d| \leq \varepsilon d, |\text{Supp}_{I,\phi}(x) - 4\eta d| \leq \varepsilon d\}.$$

Concerning $R'(I) \supseteq R(I)$ we are slightly more generous concerning the support:

$$R'(I) = R'_{\phi,\varepsilon}(I) = \{x \in V \mid |\text{Occ}_I(x) - \mu d| \leq \varepsilon d, |\text{Supp}_{I,\phi}(x) - 4\eta d| \leq 4\varepsilon d\}.$$

Given a set of variables $W \subseteq V$, the boundary of W with respect to a (random) instance I and $\varepsilon > 0$ is $\partial(W) = \partial_{I,\varepsilon}(W) = \{x \in W \mid |\{x, W, W, W\}_I| \leq (\mu - 2\varepsilon)d\}$. (Recall that $E[\text{Occ}(x)] \sim \mu d$.) The core of W with respect to I and $\varepsilon > 0$, $\mathcal{C}(W) = \mathcal{C}_{I,\varepsilon}(W)$, is the largest subset $W' \subseteq W$ with $\partial(W') = \emptyset$. It can be obtained by the following algorithm which iteratively deletes a variable from the current boundary:

$$W' := W$$

while $\partial(W') \neq \emptyset$: Pick any $x \in \partial(W')$; $W' := W' \setminus \{x\}$.

The correctness follows with the invariant $\mathcal{C}(W) \subseteq W'$.

The following algorithm to find a satisfying assignment in the not-all-equal sense yields the main result. It is inspired by the algorithm in [8] for 3-SAT.

Algorithm 10. Input: Constants d, η_i, ε and a 4-SAT formula I over Var_n (generated as above).

1. Construct the graph $G = G_I = (V, E)$ with $V = \text{Lit}_n$. For $l \neq k \in \text{Lit}_n$ we have $\{l, k\} \in E$ iff we have a clause $C \in I$ with $C = l \vee k \vee x \vee y$. (Note that we have no loops or multiple edges.)
2. Construct $G' = (V, E')$ by deleting all edges incident with vertices $l \in V$ with $d_l \geq 180d$. Here d_l is the degree of l . Compute the eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{2n}$ of the adjacency matrix A of G' . Let $e_{2n} = (a_1, \dots, a_{2n})^t$ (t for transpose) be the eigenvector of the most negative eigenvalue λ_{2n} . Let a_i be the entry corresponding to the variable x_i and a_{n+i} be the entry corresponding to $\neg x_i$.
3. We construct the assignment π : For the variable x_i we let $\pi(x_i) := 1$ if the entry $a_i > a_{n+i}$. Otherwise we set $\pi(x_i) := 0$.
4. For $i = 1, 2, \dots, \log n$ do

$$W := \{x \in V \mid |\{C \in F \mid (x \in C \text{ or } \neg x \in C) \text{ and } C \text{ false under } \pi\}| \geq 5\varepsilon d\}.$$

For all $x \in W$ do $\pi(x) := 1 - \pi(x)$.

5. We consider the core $\mathcal{C}_{I,\varepsilon}(R'_{\pi,\varepsilon}(I))$. (For $R'_{\pi,\varepsilon}(I)$ recall the definition above.) Modify π to a partial assignment by unassigning all variables not belonging to the core $\mathcal{C}' = \mathcal{C}_{I,\varepsilon}(R'_{\pi,\varepsilon}(I))$.
6. Construct the graph $\Gamma = (\text{Var}, E)$ where $\{x, y\} \in E$ iff $x, y \in V \setminus \mathcal{C}'$ and $\{x, y, -, -\}_I \neq \emptyset$. (Note that $\pi(x), \pi(y)$ is undefined at present.) Determine the connected components of Γ . If any of these has more than $\log n$ vertices the algorithm fails. Otherwise it searches for a satisfying assignment by trying out all possibilities for each connected component by itself and assigning the

unassigned variables of π such that a not-all-equal solution of I is obtained, if possible. If no such assignment is found the inconclusive answer is given.

3.2 Proof of Theorem 3

For the subsequent analysis of this algorithm we let ϕ be the assignment fixed in Step 1 of the generation algorithm. Usually we denote by I a random instance.

Theorem 11. *Let $\delta > 0$ and d be a positive integer. Then for any random instance I of size n and any assignment ϕ to the variables of I , we have*
 $|\{x \in \text{Var} \mid \pi(x) \neq \phi(x)\}| \leq \delta n$, *implies*
 $|\{x \in \text{Var} \mid \pi(x) \neq 1 - \phi(x)\}| \leq \delta n$.

(The proof is based Flaxman’s work [8] but is somehow simpler. The details can be found in [12].)

Let G' be the graph constructed from the random instance I in Step 2 of Algorithm 10. Let ϕ be the satisfying assignment we want to find. The set of literals true under ϕ are denoted by T_ϕ . Then $F_\phi = \text{Lit}_n \setminus T_\phi$ are the literals false under ϕ . We divide the adjacency matrix A of G' into 4 blocks $A_{T,T}$, $A_{F,T}$, $A_{T,F}$ and $A_{F,F}$. The block $A_{F,F}$ contains all rows and all columns of A that corresponds to literals from F_ϕ . The other blocks are defined analogously. The expected number of 1’s in $A_{F,F}$ is by the construction $2 \cdot (2d_1 + d_2)n \cdot (1 + o(1))$. In $A_{T,T}$ we can expect $2 \cdot (d_2 + 2d_3)n \cdot (1 + o(1))$ and in each of $A_{F,T}$ and $A_{T,F}$ we expect $2(d_1 + 2d_2 + d_3)n \cdot (1 + o(1))$ times a 1.

Take the all-1-vector $\mathbf{1}$ then we can expect for $\mathbf{1}^t A \mathbf{1}$ the value

$$(2 \cdot (d_2 + 2d_3) + 2 \cdot (2d_1 + d_2) + 4 \cdot (d_1 + 2d_2 + d_3))n \cdot (1 + o(1)) = (8d_1 + 12d_2 + 8d_3)n \cdot (1 + o(1)).$$

Now take the vector v of dimension $|\text{Lit}_n|$ with $v_i = 1$ if $i \in T_\phi$ and $v_i = -1$ if $i \in F_\phi$. Note, v is perpendicular to $\mathbf{1}$. The expected value of $v^t A v$ is

$$(2 \cdot (d_2 + 2d_3) + 2 \cdot (2d_1 + d_2) - 4 \cdot (d_1 + 2d_2 + d_3)) \cdot n \cdot (1 + o(1)) = -4d_2 \cdot n \cdot (1 + o(1)).$$

Note that the expectation of both $\mathbf{1}^t A \mathbf{1}$ and $v^t A v$ grows linear in d . One can show that all vectors u of length \sqrt{n} perpendicular to $\mathbf{1}$ and v have an expected value for $u^t A u$ of $O(\sqrt{d} \cdot n)$. So, the eigenvector e_{2n} belonging to the smallest eigenvalue of A must be nearly a linear combination of $\mathbf{1}$ and v . Following this idea one can show that almost all signs of v (and so almost the assignment ϕ) can be reconstructed from e_{2n} . The fraction of signs we can not reconstruct is constant but can be made arbitrarily small by increasing d . \square

The remaining part of the algorithm is also analyzed based on Flaxman’s work, but some subtle details have to be taken care of. With Theorem 11 we assume that for the assignment π after Step 3 $|\{\pi(x) = \phi(x)\}| \geq (1 - \delta)n$. If $|\{\pi(x) = 1 - \phi(x)\}| \geq (1 - \delta)n$ we proceed analogously. We pick an ε sufficiently small (for this d must be sufficiently large.)

Lemma 12.

$$C \cdot d \cdot (1 - e^{-\Omega(n)}) \leq |R_{\phi,\varepsilon}(I)| \geq (1 - e^{-d/C})n$$

$$\delta = \delta(\varepsilon) \dots \dots \dots | \{U, U, -, -\}_I | \leq 1/9 \cdot \varepsilon d |U|.$$

The correctness follows from standard calculations which can be found in [12].

Lemma 13. *I* $\dots \dots \dots$

$$|\mathcal{C}_{I,\varepsilon}(R_{\phi,\varepsilon}(I))| \geq (1 - 2^{-d/C})n.$$

Note, that the lemma means that we have $|\mathcal{C}_{I,\varepsilon}(R_{\phi,\varepsilon}(I))| \geq (1 - 2^{-d/C})n$ with probability at least $1 - O(n^{-\sqrt{d}})$.

Let $R = R_{\phi,\varepsilon}(I)$ and $\mathcal{C} = \mathcal{C}_{I,\varepsilon}(R)$ and recall the algorithm from Subsection 3.1 to generate \mathcal{C} . We show that the while loop of this algorithm is executed $m \leq e^{-d/C}n$ -times. Then the result follows, for $|\text{Var} \setminus \mathcal{C}| = |\text{Var} \setminus R| + m \leq 2e^{-d/C}n \leq 2^{-d/C}n$.

Assume that the loop of the algorithm is executed at least m -times and consider the first m executions of the loop. (We specify m further below.) Let $x_i = x$ after the i 'th execution of the loop and let $\mathcal{C}_0 = R$ and $\mathcal{C}_i = \mathcal{C}_{i-1} \setminus \{x_i\}$. Then \mathcal{C}_i is the value of W' of the algorithm after the i 'th execution of the while loop. Let $U_i = \text{Var} \setminus \mathcal{C}_i$. As $x_i \in \partial(\mathcal{C}_{i-1}) \subseteq R$ we have that $|\{x_i, -, -, -\}_I| \geq (\mu - \varepsilon)d$. As $x_i \in \partial(\mathcal{C}_{i-1})$ we have that $|\{x_i, \mathcal{C}_{i-1}, \mathcal{C}_{i-1}, \mathcal{C}_{i-1}\}_I| \leq (\mu - 2\varepsilon)d$. Therefore $|\{x_i, U_{i-1}, -, -\}_I| \geq \varepsilon d$ and thus $\sum_{i=1}^m |\{x_i, U_{i-1}, -, -\}_I| \geq m\varepsilon d$. Clauses from $\{x_i, x_{i+1}, x_{i+2}, U_{i-1}\}_I$ are counted 3-times in the sum. No clause is counted 4 or more times. Thus the number of different clauses contributing to the sum is $\geq 1/3m\varepsilon d$. As for all i $\{x_i, U_{i-1}, -, -\}_I \subseteq \{U_m, U_m, -, -\}_I$ we get that $|\{U_m, U_m, -, -\}_I| \geq 1/3m\varepsilon d$. Now assuming $m = e^{-d/C}n$ we have that $|U_m| = 2m \leq 2\delta n$ and $|\{U_m, U_m, -, -\}_I| \geq 1/6|U_m|\varepsilon d$ contradicting item 2 of Lemma 12. □

Lemma 14. $\mathcal{C} = \mathcal{C}_{I,\varepsilon}(R_{\phi,\varepsilon}(I))$, $\pi_i = \dots \dots \dots \pi \dots \dots \dots i'$. $B_i = \{x \in \mathcal{C} \mid \pi_i(x) \neq \phi(x)\}$ $I \dots \dots \dots i \leq \log n \dots \dots \dots |B_i| \leq |B_{i-1}|/2$.

This lemma directly implies that after Step 4 all variables from \mathcal{C} have the right truth value.

Corollary 15. $\dots \dots \dots \mathcal{C} \dots \dots \dots \mathcal{C} \subseteq \{x \in \text{Var}_n \mid \pi(x) = \phi(x)\}$.

From Theorem 11 we know that $|B_0| \leq \delta n$. Further below we show that for all $x \in B_i$ we have $|\{x, B_{i-1}, -, -\}_I| \geq 2\varepsilon d$. This implies the claim as follows. By induction we can assume that $|B_{i-1}| \leq \delta n$. Assuming that $|B_i| > |B_{i-1}|/2$ we let $B' \subseteq B_i$ with $|B'| = \lfloor |B_{i-1}|/2 \rfloor + 1$. From the statement above we get that $\sum_{x \in B'} |\{x, B_{i-1}, -, -\}_I| \geq |B'|2\varepsilon d$. For $x_1, \dots, x_4 \in B' \cap B_{i-1}$ all distinct a clause like $x_1 \vee x_2 \vee x_3 \vee x_4$ is counted 4-times. No clause is

counted more than 4-times. This implies that $|\{B', B_{i-1}, -, -\}_I| \geq |B'|2\epsilon d/4$. Now consider $B = B' \cup B_{i-1}$, then we have that $|B| \leq 2\delta n$, but $|\{B, B, -, -\}_I| \geq |\{B', B_{i-1}, -, -\}_I| \geq |B|2\epsilon d/16$ as $|B'| \geq |B|/4$ in contradiction to item 2 of Lemma 12.

We now show the statement above by the case distinction that for $x \in B_i$ either $x \in B_{i-1}$ or $x \notin B_{i-1}$. Let

$$a = |\{C \in F \mid (x \in C \text{ or } \neg x \in C) \text{ and } C \text{ false under } \pi_i\}|.$$

If $x \in B_i$ and $x \in B_{i-1}$ we have that $\pi_{i-1}(x) = \pi_i(x) = \neg\phi(x)$. Moreover, as the value of x has not been changed by the loop we know that $a \leq 5\epsilon d$. As $x \in \mathcal{C} \subseteq R(I)$ we have that $\text{Supp}_{I,\phi}(x) \geq (4\eta - \epsilon)d$. Therefore we have at least $(4\eta - \epsilon)d - 5\epsilon d = (4\eta - 6\epsilon)d$ clauses $C \in F$ with the property: There is a literal $l \in C$ which makes C true under π_{i-1} , but for the underlying variable y we have that $\pi_{i-1}(y) \neq \phi(y)$. As $x \in \mathcal{C}$ we have that $|\{x, -, -, -\}_I| \leq (\mu + \epsilon)d$ and $|\{x, \mathcal{C}, \mathcal{C}, \mathcal{C}\}_I| \geq (\mu - 2\epsilon)d$. Therefore $|\{x, \text{Var} \setminus \mathcal{C}, -, -\}_I| \leq 3\epsilon d$. Hence, among the $(4\eta - 6\epsilon)d$ clauses containing x above, we have at least $(4\eta - 6\epsilon)d - 3\epsilon d = (4\eta - 9\epsilon)d \geq 2\epsilon d$ (ϵ sufficiently small) clauses which contain a literal over a variable y from C which is false under π_{i-1} . For this y we clearly have $y \in B_{i-1}$.

If $x \in B_i$ and $x \notin B_{i-1}$ the value of x has been changed in the loop of the algorithm and we know that $a \geq 5\epsilon d$. Each of these a clauses obviously contains a literal over a variable y such that $\pi_{i-1}(y) = \neg\phi(y)$. We show that for at least $2\epsilon d$ of these a clauses we have that $y \in B_{i-1}$. This follows as $|\{x, -, -, -\}_I| \leq (\mu + \epsilon)d$ and $|\{x, \mathcal{C}, \mathcal{C}, \mathcal{C}\}_I| \geq (\mu - 2\epsilon)d$. Therefore $|\{x, \text{Var} \setminus \mathcal{C}, -, -\}_I| \leq 3\epsilon d$ and we get $|\{x, B_{i-1}, -, -, -\}_I| \geq 2\epsilon d$. □

The following lemma implies that after Step 5 of the algorithm the core \mathcal{C} remains correctly assigned. Its proof can be found in [12].

Lemma 16. π
 I
 $\mathcal{C} = \mathcal{C}_{I,\epsilon}(R_{\phi,\epsilon}(I)) \subseteq \{x \mid \pi(x) \text{ defined}\}.$
 $\forall x \in \mathcal{C} \quad \pi(x) = \phi(x).$

Each connected component of Γ of size $\geq \log n$ contains a connected component of size $\log n$. We show that the expected value of such components goes to 0. To this end let $T' = (V(T'), E(T'))$ be a fixed tree (connected graph without cycles) with $V(T') \subseteq \text{Var}$ and $|V(T')| = \log n$. Let $T \subseteq CT_{nae,\phi}$ be a fixed set of 4-clauses such that for each $\{x, y\} \in E(T')$ we have a clause $C \in \{x, y, -, -\}_T$. Let T be a minimal set with this property. The tree T' induced by T occurs only then in Γ if firstly $V(T') \cap \mathcal{C}_I(R(I)) = \emptyset$ and secondly $T \subseteq I$. Thus we have to bound

$$\Pr[T \subseteq I \text{ and } V(T') \cap \mathcal{C}_I(R(I))] = \Pr[T \subseteq I] \cdot \Pr[V(T') \cap \mathcal{C}_I(R(I)) \mid T \subseteq I].$$

Lemma 17. Let T and T' be trees with $V(T) \cap V(T') = \emptyset$.

$$\Pr[T \subseteq I] \leq (d/n^3)^{|T|}$$

$$\Pr[V(T') \cap \mathcal{C}_I(R(I)) \mid T \subseteq I] = O\left(n^{-\sqrt{d}}\right)$$

The first item is easy to see as each of the $|T|$ clauses is chosen with probability at most d/n^3 . The second one is more difficult to show – not only because of the condition $T \subseteq I$. We omit the proof and refer to [12].

Lemma 18. Let Γ be a graph with n vertices and $\log n$ edges.

We bound the probability that Γ has a connected component by the expected value of such components. This expectation can be bounded above by

$$\sum_{T, T'} \Pr[T \subseteq I \text{ and } V(T') \cap \mathcal{C} = \emptyset] \leq \sum_{T, T'} \left(\frac{d}{n^3}\right)^{|T|} \cdot O(n^{-\sqrt{d}}) \tag{5}$$

where the sums goes over all trees T' with $V(T') \subseteq \text{Var}$ and all minimal sets of clauses T so that for any edge $\{x, y\} \in E(T')$ we have a clause in $\{x, y, -, -\}_T$.

By an appropriate calculation of the number of such pairs T and T' one can show that (5) is bounded above by $n^{O(\log d)} \cdot O(n^{-\sqrt{d}})$, cf. [12]. This bound is $o(1)$ for d large enough but still constant and we are done. \square

References

1. Alon N., Feige U., Widgerson A., Zuckerman D.: Derandomized Graph Products. Computational Complexity 5 (1995), 60–75.
2. Alon N., Kahale N.: A spectral technique for coloring random 3-colourable graphs. DIMACS TR 94-35, 1994.
3. Chen H., Frieze A.: Coloring bipartite hypergraphs. Proc. 5th IPCO 1996, LNCS, 345–358.
4. Coja-Oghlan A., Goerdt A., Lanka A.: Strong Refutation Heuristics for Random k -SAT. RANDOM 2004. To appear. (<http://www.tu-chemnitz.de/informatik/HomePages/TI/publikationen.php>)
5. Coja-Oghlan A., Goerdt A., Lanka A., Schädlich F.: Techniques from combinatorial approximation algorithms yield efficient algorithms for random 2k-SAT. Theoretical Computer Science. To appear.
6. Feige U.: Relations between average case complexity and approximation complexity. Proc. 24th STOC (2002), 534–543.
7. Feige U., Ofek E.: Easily refutable subformulas of large random 3CNF formulas. (<http://www.wisdom.weizmann.ac.il/~erano/>)
8. Flaxman A.: A spectral technique for random satisfiable 3CNF formulas. Proc. SoDA 2002, SIAM.
9. Garey M.R., Johnson D.S.: Computers and Intractability. 1979.
10. Goerdt, A., Jurdzinski, T.: Some results on random unsatisfiable k -SAT instances and approximation algorithms applied to random structures. Combinatorics, Probability and Computing 12 (2003) 245–267

11. Goerdts A., Lanka A.: Recognizing more random unsatisfiable 3-SAT instances efficiently. Proc. Typical Case Complexity and Phase Transitions, Satellite Workshop of Logic in Computer Science 2003 (Ottawa). To appear.
12. Goerdts A., Lanka A.: The algorithmic hardness and easiness of random 4-SAT formulas. Technical report. (<http://www.tu-chemnitz.de/informatik/HomePages/TI/publikationen.php>)
13. Håstad J.: Clique is Hard to Approximate within $n^{1-\epsilon}$. Acta Mathematica 182 (1999) 105–142
14. Janson S., Luczak T., Ruciński A.: Random graphs. John Wiley and Sons 2000.
15. Lubotsky A., Phillips R., Sarnak P.: Ramanujan Graphs. Combinatorica 8 (3), 1988 261–277
16. Peeters R.: The maximum edge biclique problem is \mathcal{NP} -complete. Research Memorandum 789, Faculty of Economics and Business Administration, Tilburg University, 2000. (<http://econpapers.hhs.se/paper/dgrkubrem/2000789.htm>)
17. Strang G.: Linear Algebra and its Applications. Harcourt Brace Jovanovich. 1988.

Minimum Common String Partition Problem: Hardness and Approximations

Avraham Goldstein¹, Petr Kolman^{2,*}, and Jie Zheng^{3,**}

¹ avi_goldstein@netzero.net

² Institute for Theoretical Computer Science, Charles University, Malostranské nám.
25, 118 00 Praha 1, Czech Republic

kolman@kam.mff.cuni.cz

³ Department of Computer Science, University of California, Riverside, CA 92521
zjie@cs.ucr.edu

Abstract. String comparison is a fundamental problem in computer science, with applications in areas such as computational biology, text processing or compression. In this paper we address the minimum common string partition problem, a string comparison problem with tight connection to the problem of sorting by reversals with duplicates, a key problem in genome rearrangement.

A *partition* of a string A is a sequence $\mathcal{P} = (P_1, P_2, \dots, P_m)$ of strings, called the *blocks*, whose concatenation is equal to A . Given a partition \mathcal{P} of a string A and a partition \mathcal{Q} of a string B , we say that the pair $\langle \mathcal{P}, \mathcal{Q} \rangle$ is a *common partition* of A and B if \mathcal{Q} is a permutation of \mathcal{P} . The *minimum common string partition* problem (MCSP) is to find a common partition of two strings A and B with the minimum number of blocks. The restricted version of MCSP where each letter occurs at most k times in each input string, is denoted by k -MCSP.

In this paper, we show that 2-MCSP (and therefore MCSP) is NP-hard and, moreover, even APX-hard. We describe a 1.1037-approximation for 2-MCSP and a linear time 4-approximation algorithm for 3-MCSP. We are not aware of any better approximations.

1 Introduction

String comparison is a fundamental problem in computer science, with applications in areas such as computational biology, text processing or compression. Typically, a set of string operations is given (e.g., delete, insert and change a character, move a substring or reverse a substring) and the task is to find the minimum number of operations needed to convert one string to the other. Edit distance or permutation sorting by reversals are two well known examples. In this paper we address, motivated mainly by genome rearrangement applications, the

* Research done while visiting University of California at Riverside. Partially supported by project LN00A056 of MŠMT ČR, and NSF grants CCR-0208856 and ACI-0085910.

** Supported by NSF grant DBI-0321756.

minimum common string partition problem (MCSP). Though MCSP takes a static approach to string comparison, it has tight connection to the problem of sorting by reversals with duplicates, a key problem in genome rearrangement.

A *partition* of a string A is a sequence $\mathcal{P} = (P_1, P_2, \dots, P_m)$ of strings whose concatenation is equal to A , that is $P_1P_2 \dots P_m = A$. The strings P_i are called the *blocks* of \mathcal{P} . Given a partition \mathcal{P} of a string A and a partition \mathcal{Q} of a string B , we say that the pair $\pi = \langle \mathcal{P}, \mathcal{Q} \rangle$ is a *common partition* of A and B if \mathcal{Q} is a permutation of \mathcal{P} . The *minimum common string partition problem* is to find a common partition of A, B with the minimum number of blocks. The restricted version of MCSP where each letter occurs at most k times in each input string, is denoted by k -MCSP. We denote by $\#\pi$ the number of blocks in a common partition π . We say that two strings A and B are *equi-occurring* if every letter appears the same number of times in A and B .

In this paper, we show that 2-MCSP (and therefore MCSP) is NP-hard and, moreover, even APX-hard. We also describe a 1.1037-approximation for 2-MCSP and a linear time 4-approximation algorithm for 3-MCSP. We are not aware of any better approximations. For the lack of space, we have to omit several proofs; they will appear in the full version of the paper.

The *signed minimum common string partition problem* (SMCSP) is a variant of MCSP in which each letter of the two input strings is given a “+” or “-” sign. For a string P with signs, let $-P$ denote the reverse of P , with each sign flipped. A common partition of two signed strings A and B is the pair $\pi = \langle \mathcal{P}, \mathcal{Q} \rangle$ of a partition $\mathcal{P} = (P_1, P_2, \dots, P_m)$ of A and a partition $\mathcal{Q} = (Q_1, Q_2, \dots, Q_m)$ of B together with a permutation σ on $[m]$ such that for each $i \in [m]$, either $P_i = Q_{\sigma(i)}$, or $P_i = -Q_{\sigma(i)}$. All of our results apply also to signed MCSP.

1-MCSP coincides with the breakpoint distance problem of two permutations [12] which is to count the number of pairs of symbols that are adjacent in the first string but not in the other; this problem is obviously solvable in polynomial time. Similarly as the breakpoint distance problem does, most of the rearrangement literature works with the assumption that a genome contains only one copy of each gene. Under this assumption, a lot of attention was given to the problem of sorting by reversals which is solvable in polynomial time for strings with signs [9] but is NP-hard for strings without signs [3]. The assumption about uniqueness of each gene is unwarranted for genomes with multi-gene families such as the human genome [10]. Chen et al. [4] studied a generalization of the problem, the problem of signed reversal distance with duplicates (SRDD); according to them, SRDD is NP-hard even if there are at most two copies of each gene. They also introduced the signed minimum common partition problem as a tool for dealing with SRDD. Chen et al. observe that for any two related signed strings A and B , the size of a minimum common partition and the minimum number of reversal operations needed to transform A to B , are within a multiplicative factor 2 of each other. (In the case of unsigned strings, no similar relation holds: the reversal distance of $A = 1234 \dots n$ and $B = n \dots 4321$ is 1 while the size of minimum common partition is $n - 1$.) They also give a 1.5-approximation algorithm for 2-MCSP. Christie and Irving [5] consider the

problem of (unsigned) reversal distance with duplicates (RDD) and prove that it is NP-hard even for strings over binary alphabet.

Chrobak et al. [6] analyze a natural heuristic for MCSP, the greedy⁴ algorithm: iteratively, at each step extract a longest common substring from the input strings. They show that for 2-MCSP, the approximation ratio is exactly 3, for 4-MCSP the approximation ratio is $\Omega(\log n)$; for the general MCSP, the approximation ratio is between $\Omega(n^{0.43})$ and $O(n^{0.67})$. The same bounds apply for SMCSP.

Closely related is the problem of edit distance with moves in which the allowed string operations are the following: insert a character, delete a character, move a substring. Cormode and Muthukrishnan [7] describe an $O(\log n \log^* n)$ -approximation algorithm for this problem. Shapira and Storer [11] observed that restriction to move-a-substring operations only (instead of allowing all three operations listed above) does not effect the edit-distance of two strings by more than a constant multiplicative factor. Since the size of a minimum common partition of two strings and their distance with respect to move-a-substring operations differ only by a constant multiplicative factor, the algorithm of Cormode and Muthukrishnan yields an $O(\log n \log^* n)$ -approximation for MCSP.

1.1 Preliminaries

Throughout the paper, we assume that the two strings A, B given as input to MCSP are related. This is a necessary and sufficient condition for the existence of a common partition.

Given a string $A = a_1 \dots a_n$, for the sake of simplicity we will use the symbol a_i to denote two different things. First, a_i may denote the specific occurrence of the letter a_i in the string A , namely the occurrence on position i . Alternatively, a_i may denote just the letter itself, without any relation to the string A . Which alternative we mean will be clear from context.

Let Σ denote the set of all letters that occur in A . A duo is an ordered pair of letters $xy \in \Sigma^2$ that occur consecutively in A or B (that is, there exists an i such that $x = a_i$ and $y = a_{i+1}$, or $x = b_i$ and $y = b_{i+1}$). A *specific duo* is an occurrence of a duo in A or B . The difference is that a duo is just a pair of letters whereas a specific duo is a pair of letters together with its position. A *match* is a pair $(a_i a_{i+1}, b_j b_{j+1})$ of specific duos, one from A and the other one from B , such that $a_i = b_j$ and $a_{i+1} = b_{j+1}$. Two matches $(a_i a_{i+1}, b_j b_{j+1})$ and $(a_k a_{k+1}, b_l b_{l+1})$, $i \leq k$, are *disjoint* if either $i = k$ and $j \neq l$, or $i + 1 = k$ and $j + 1 \neq l$, or $i + 1 < k$ and $\{j, j + 1\} \cap \{l, l + 1\} = \emptyset$.

We construct a graph $G = (V, E)$ of A and B as follows. The set of nodes V consists of all matches of A and B and the set of edges E consists of

⁴ Shapira and Storer [11] also analyzed the greedy algorithm and claimed an $O(\log n)$ bound on its approximation ratio; unfortunately, the analysis was flawed, it applies only to a special subclass of MCSP problems.

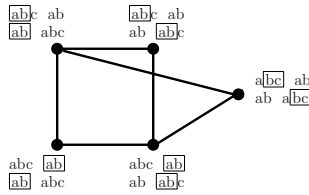


Fig. 1. Conflict graph for MCSP instance $A = abcab$ and $B = ababc$

all pairs of matches that are in conflict. Figure 1 shows an example of a conflict graph. The number of vertices in G can be much higher than the length of the strings A and B (and is trivially bounded by n^2).

Lemma 1. $A = a_1 \dots a_n$ $B = b_1 \dots b_n$ $MIS(G)$ G A B m
 $n - MIS(G) = m$

Given an optimal solution for MCSP, let S be the set of all matches that are used in this solution. Clearly, S is an independent set in G and $|S| = n - 1 - (m - 1)$.

Conversely, given a maximum independent set S , we cut the string A between a_i and a_{i+1} for every specific duo $a_i a_{i+1}$ that does not appear in any match in S , and similarly for B . In this way, $n - 1 - |S|$ duos are cut in A and also in B , resulting in $n - |S|$ blocks of A and $n - |S|$ blocks of B . Clearly, the blocks from A can be matched with the blocks from B , and therefore $m \leq n - |S|$. \square

Maximum independent set is an NP-hard problem, yet, two approximation algorithms for MCSP described in this paper make use of this reduction.

MCSP for multisets of strings. Instead of two strings A, B , there are two multisets \mathcal{A}, \mathcal{B} of strings on input. Similarly as before, a partition of the multiset $\mathcal{A} = \{A_1, \dots, A_l\}$ is a sequence of strings $A_{1,1}, \dots, A_{1,k_1}, A_{2,1}, \dots, A_{2,k_2}, \dots, A_{l,1}, \dots, A_{l,k_l}$, such that $A_i = A_{i,1} \dots A_{i,k_i}$ for $i \in [l]$. For two multisets of strings, the common partition, the minimum common partition and the related-relation are defined similarly as for pairs of strings.

Let $\mathcal{A} = \{A_1, \dots, A_l\}$ and $\mathcal{B} = \{B_1, \dots, B_h\}$ with $h \leq l$, be two related multisets of strings, and let $x_1, y_1, \dots, x_{l-1}, y_{l-1}$ be $2l - 2$ different letters that do not appear in \mathcal{A} and \mathcal{B} . Considering two strings

$$\begin{aligned} A &= A_1 x_1 y_1 A_2 x_2 y_2 A_3 \dots x_{l-1} y_{l-1} A_l, \\ B &= B_1 y_1 x_1 B_2 y_2 x_2 B_3 \dots y_{h-1} x_{h-1} B_h y_h x_h \dots y_{l-1} x_{l-1}, \end{aligned} \tag{1}$$

it is easy to see that an optimal solution for the classical MCSP instance A, B yields an optimal solution for the instance \mathcal{A}, \mathcal{B} of the multiset version, and vice versa. In particular, if m' denotes the size of a MCSP of the two multisets of strings \mathcal{A} and \mathcal{B} , and m denotes the size of a MCSP of the two strings A and B defined as above, then $m = m' + 2(l - 1)$. Thus, if one of the variants of the problems is NP-hard, so is the other.

2 Hardness of Approximation

Theorem 1. 2 MCSP \dots 2 SMCSF

We start by proving a weaker result.

Theorem 2. 2 MCSP \dots 2 SMCSF

Since an instance of MCSP can be interpreted as an instance of SMCSF with all signs positive, and since a solution of SMCSF with all signs positive can be interpreted as a solution of the original MCSP and vice versa, it is sufficient to prove the theorems for MCSP only.

The proof is by reduction from the maximum independent set problem on cubic graphs (3-MIS) [8]. Given a cubic graph $G = (V, E)$ as an input for 3-MIS, for each vertex $v \in V$ we create a small instance I_v of 2-MCSP. Then we process the edges of G one after another, and, for each edge $(u, v) \in E$, we locally modify the two small instances I_u, I_v . The final instance of 2-MCSP, denoted by I_G , is the union of all the small (modified) instances I_v . We will show that a minimum common partition of I_G yields easily a maximum independent set in G .

The small instance $I_u = (X_u, Y_u)$ for a vertex $u \in V$ is defined as follows (cf. Figure 2):

$$\begin{aligned} X_u &= \{d_u, a_u b_u, c_u d_u e_u, b_u e_u f_u g_u, f_u h_u k_u, g_u l_u, h_u\} \\ Y_u &= \{b_u, c_u d_u, a_u b_u e_u, d_u e_u f_u h_u, f_u g_u l_u, h_u k_u, g_u\} \end{aligned} \tag{2}$$

where $a_u, b_u, \dots, l_u, a_v, b_v, \dots, l_v$ are distinct letters in the alphabet. It is easy to check that I_u has a unique minimum common partition, denoted by O_u , namely:

$$\begin{aligned} O_u &= \langle (d_u, a_u b_u, c_u d_u, e_u, b_u, e_u f_u, g_u, f_u, h_u k_u, g_u l_u, h_u) \\ &\quad (b_u, c_u d_u, a_u b_u, e_u, d_u, e_u f_u, h_u, f_u, g_u l_u, h_u k_u, g_u) \rangle \end{aligned}$$

We observe that for $X_G = \bigcup_{u \in V} X_u$ and $Y_G = \bigcup_{u \in V} Y_u$, $I_G = (X_G, Y_G)$ is an instance of 2-MCSP, and the superposition of all O_u 's is a minimum common partition of I_G . For the sake of simplicity, we will sometimes abuse the notation by writing $I_G = \bigcup_{u \in V} I_u$.

The main idea of the construction is to modify the instances I_u , such that for every edge $(u, v) \in E$, a minimum common partition of $I_G = \bigcup_{u \in V} I_u$ coincides with at most one of the minimum common partitions of I_u and I_v .

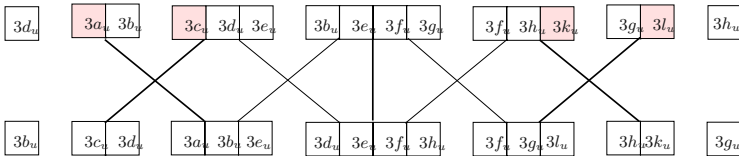


Fig. 2. An instance I_u : the lines represent all matches, with the bold lines corresponding to the matches in the minimum common partition O_u

This property will make it possible to obtain a close correspondence between maximum independent sets in G and minimum common partitions of I_G : if O_v denotes a minimum common partition of (the modified) I_v and O'_v denotes the common partition of (the modified) I_v derived from a given minimum common partition of I_G , then $U = \{u \in V \mid O'_u = O_u\}$ will be a maximum independent set of G . To avoid the need to use different indices, we use I_G to denote $\bigcup_{u \in V} I_u$ after any number of the local modifications; it will always be clear from context to which one are we referring.

For description of the modifications, a few terms will be needed. The letters a_u and c_u in X_u are called *right* and *left* letters of I_u and the letters k_u and l_u in X_u are called *right* and *left* sockets of I_u . We observe that all the four letters a_u, c_u, k_u, l_u appears only once in X_G (and once in Y_G). Given two small instances I_u and I_v and a socket s_u of I_u and a socket s_v of I_v , we say that the two sockets s_u and s_v are *compatible*, if one of them is a left socket and the other one is a right socket. Initially, all sockets are *free*.

For technical reasons, we orient the edges of G in such a way that each vertex has at most two incoming edges and at most two outgoing edges. This can be done as follows: find a maximal set (with respect to inclusion) of edge-disjoint cycles in G , and in each cycle, orient the edges to form a directed cycle. The remaining edges form a forest. For each tree in the forest, choose one of its nodes of degree one to be the root, and orient all edges in the tree away from the root. This orientation will clearly satisfy the desired properties.

We are ready to describe the local modifications. Consider an edge $\overrightarrow{(u, v)} \in E$ and a free right socket s_u of I_u and a free left socket s_v of I_v . That is, $Rs_u \in X_u$ and $s_vS \in X_v$, for some strings R and S . We modify the instances $I_u = (X_u, Y_u)$ and $I_v = (X_v, Y_v)$ as follows

$$\begin{aligned} X_u &\leftarrow X_u \cup \{Rs_uS\} - \{Rs_u\}, & X_v &\leftarrow X_v \cup \{s_u\} - \{s_vS\}, \\ Y_u &\leftarrow Y_u, & Y_v &\leftarrow Y_v \text{ with } s_v \text{ renamed by } s_u. \end{aligned} \tag{3}$$

(the symbols \cup and $-$ denote multiset operations).

After this operation, we say that the right socket s_u of I_u and the left socket s_v of I_v are *matched* (not free). Note that in Y_v , the letter s_v is renamed to s_u . All other sockets of I_u and all other sockets of I_v that were free before the operation remain free. We also note that I_u and I_v are not 2-MCSP instances. However, for every letter, the number of its occurrences is the same in X_G and in Y_G , namely at most two. Thus, I_G is still a 2-MCSP instance.

The complete reduction from a cubic graph $G = (V, E)$ to a 2-MCSP instance is done by performing the local modifications (3) for all edges in G . Since the in-degree and the out-degree of every node is bounded by two, and since every instance I_u has initially two right and two left sockets, there will always be the required free sockets.

It remains to prove that a minimum common partition for the final I_G (that is, when modifications for all edges are done) can be used to find a maximum independent set in G .

Lemma 2. Let G be a graph with N vertices and l edges. Let I_G be a graph with $9N$ vertices and $12N - l$ edges.

Let G_C be the conflict graph of I_G ; G_C has $9N$ vertices. The crucial observation is that each small instance I_u can choose independently on all other small instances four of its nine possible matches in such a way that all these $4N$ matches form an independent set in G_C (in Figure 2, these four matches are represented by the thin lines). Let O'_u denote the four matches chosen by u .

Given an independent set I of G , construct a common partition of I_G as follows. For $u \in I$, use the five matches from O_u , and for $u \notin I$, use the four matches from O'_u . The resulting solution will use $5l + 4(N - l)$ matches which corresponds to $9N - (5l + 4(N - l)) = 5N - l$ new breaks and $7N + 9N - (5l + 4(N - l)) = 12N - l$ blocks.

Conversely, given a common partition of I_G of size m , let I consist of all vertices u such that I_u contributes 5 matches (i.e., 11 blocks) to the common partition. Then, $l \geq 12N - m$, and the proof is completed. \square

Since the reduction can clearly be done in polynomial time (even in linear), with respect to $n = |V|$ and $m = |E|$, the proof of NP-hardness is completed.

The proof of APX-hardness relies on the same reduction. Berman and Karpinski [2] proved that it is NP-hard to approximate 3-MIS within $\frac{140}{139} - \epsilon$, for every $\epsilon > 0$. The relation between the size of an independent set and the size of a common partition in our reduction, given in Lemma 2, implies therefore that 2-MCSP for multisets of string is also APX-hard problem. Transforming the multiset MCSP instance into an instance with just two strings affects only the factor of inapproximability.

3 Algorithms

3.1 2-MCSP Reduces to MIN 2-SAT

Theorem 3. For any $\alpha > 1$, there is a polynomial time algorithm that approximates 2-MCSP within a factor of α by solving MIN 2-SAT within a factor of α .

Plugging in a recent 1.1037-approximation algorithm of Avidor and Zwick [1], we get the following result.

Corollary 1. There is a polynomial time algorithm that approximates 2-MCSP within a factor of 1.1037 by solving MIN 2-SAT within a factor of 2.

(Theorem 3)

Let A and B be two related (unsigned) strings. We start the proof with two assumptions that will simplify the presentation:

- (1) no duo appears at the same time twice in A and twice in B , and that
- (2) every letter appears exactly twice in both strings.

Concerning the first assumption, the point is that in 2-MCSP, the minimum common partition never has to break such a duo. Thus, if there exists in A and B such a duo, it is possible to replace it by a new letter, solve the modified instance and then replace the new letter back by the original duo. Concerning the other, a letter that appears only once can be replaced by two copies of itself. A minimum common partition never has to use a break between these two copies, so they can be easily replaced back to a single letter, when the solution for the modified instance is found.

The main idea of the reduction is to represent a common partition of A and B as a truth assignment of a (properly chosen) set of binary variables. With each letter $a \in \Sigma$ we associate a binary variable X_a . For each letter $a \in \Sigma$, there are exactly two ways to map the two occurrences of a in A onto the two occurrences of a in B : either the first a from A is mapped on the first a in B and the second a from A on the second a in B , or the other way round. In the first case, we say that a is mapped straight, and in the other case that a is mapped across. Given a common partition π of A and B , if a letter $a \in \Sigma$ is mapped straight we set $X_a = 1$, and if a is mapped across we set $X_a = 0$. In this way, every common partition can be turned into truth assignment of the variables X_a , $a \in \Sigma$, and vice versa. Thus, there is one-to-one correspondence between truth-assignments for the variables X_a , $a \in \Sigma$, and common partitions (viewed as mappings) of A and B .

With this correspondence between truth assignments and common partitions, our next goal is to transform the two input strings A and B into a boolean formula φ such that

- φ is a conjunction of disjunctions (OR) and exclusive disjunctions (XOR),
- each clause contains at most two literals, and
- the minimal number of satisfied clauses in φ is equal to the number of breaks in a minimum common partition of A and B .

The formula φ consists of $n - 1$ clauses, with a clause C_i for each specific duo $a_i a_{i+1}$, $i \in [n - 1]$. For $i \in [n - 1]$, let $s_i = 1$ if a_i is the first occurrence of the letter a_i in A (that is, the other copy of the same letter occurs on a position $i' > i$), and let $s_i = 2$ otherwise (that is, if a_i is the second occurrence of the letter a_i in A). Similarly, let $t_i = 1$ if b_i is the first occurrence of the letter b_i in B and let $t_i = 2$ otherwise. We are ready to define φ . There will be three types of clauses in φ .

If the duo $a_i a_{i+1}$ does not appear in B at all, we define $C_i = 1$. Let b be the number of clauses of this type.

If the duo $a_i a_{i+1}$ appears once in B , say as $b_j b_{j+1}$, let $Y = X_i$ if $s_i \neq t_j$, and let $Y = \neg X_i$ otherwise; similarly, let $Z = X_{i+1}$ if $s_{i+1} \neq t_{j+1}$ and let $Z = \neg X_{i+1}$ otherwise. We define $C_i = Y \vee Z$. In this way, the clause C_i is satisfied if and only if $i, i + 1$ is a break in a common partition consistent with the truth assignment of X_i and X_{i+1} .

Similarly, if the duo $a_i a_{i+1}$ appears twice in B , we set $C_i = X_i \oplus X_{i+1}$ if $s_i = s_{i+1}$, and we set $C_i = \neg X_i \oplus X_{i+1}$ otherwise, where \oplus denotes the exclusive disjunction. Again, the clause C_i is satisfied if and only if $i, i + 1$ is a break in a

common partition consistent with the truth assignment of X_i and X_{i+1} . Let k denote the number of these clauses.

By the construction, a truth assignment that satisfies the minimum number of clauses in $\varphi = C_1 \wedge \dots \wedge C_{n-1}$ corresponds to a minimum common partition of A and B . In particular, the number of satisfied clauses is equal to the number of breaks in the common partition which is by one smaller than the number of blocks in the partition.

The formula φ resembles an instance of 2-SAT. However, 2-SAT formulas do not allow XOR clauses. One way to get around this is to replace every XOR clause by two OR clauses. This increases the length of the formula which in turn increases the resulting approximation ratio for 2-MCSP. To avoid this drawback, we observe that for each XOR clause there is a unique inherent break. For the lack of space, we omit the details how this yields the α -approximation. \square

3.2 Linear Time 4-Approximation for 3-MCSP

In this section we exploit again the relation of MCSP and MIS in the conflict graph. The main idea of the algorithm is to cut the strings A and B into few pieces in such a way that the conflict graph of the modified instance becomes simple, making it possible to find MIS in polynomial time. Similarly as in Section 3.1 we assume, without loss of generality, that no duo has three occurrences in A and in B at the same time. We augment both strings by a new character $a_{n+1} = b_{n+1} = \$$.

A duo ab is *ambiguous* if the number of its occurrences in A equals the number of its occurrences in B , and is *unique* otherwise. As before, let m denote the size of a minimum common partition of a given pair of strings A and B .

Observation 4 *Let A and B be strings over an alphabet Σ . Let $a \in \Sigma$ be a character that occurs in both strings. Let ab be a duo in A and B . Let i be the index of the first occurrence of a in A and j be the index of the first occurrence of a in B . Then, ab is ambiguous if and only if $i = j$.*

In the first phase of the algorithm, for every bad duo ab , we cut both strings A and B after the i -th occurrence of a . We charge the cuts of ab to the breaks that appear by Observation 4 in the optimal partition, that is, to the breaks after letter a . At most three cuts are charged to a single break. Let \mathcal{A} and \mathcal{B} denote the two multisets of strings we obtain from A and B after performing all these cuts.

At this point, every specific duo of \mathcal{A} and \mathcal{B} has either one or two matches. In the first case, we talk about a *unique match* and a *unique duo*, in the later case about an *ambiguous match* and an *ambiguous duo*. There are four vertices in the conflict graph corresponding to matches of an ambiguous duo and they are connected in a 4-cycle. The 4-cycle will be called a *bad cycle* ab ; a vertex corresponding to a match of a unique duo will be called a *good vertex*. We say that two duos ab and cd are *in conflict* if a match of the first duo is in a conflict with a match of the other duo.

Let G_1 be the conflict graph of \mathcal{A} and \mathcal{B} . The edges of G_1 can be classified into three groups: edges between two ambiguous matches, between an ambiguous

match and a unique match, and, between two unique matches. We are going to have a closer look on these three cases.

Consider two ambiguous duos that interfere, say ab and bc . There are only two ways how the two occurrences of ab and the two occurrences of bc can appear in \mathcal{A} : either there are two occurrences of a substring abc , or a single occurrence of each of bc , abc and ab , in any order. There are the same possibilities for \mathcal{B} . Thus, there are only three basic ways how the duos ab and bc can appear in the strings \mathcal{A} and \mathcal{B} (up to symmetry of \mathcal{A} and \mathcal{B} and up to permutation of the depicted substrings):

$$\mathcal{A} = \dots abc \dots abc \dots, \quad \mathcal{B} = \dots abc \dots abc \dots \quad (1.1)$$

$$\mathcal{A} = \dots bc \dots abc \dots ab \dots, \quad \mathcal{B} = \dots bc \dots abc \dots ab \dots \quad (1.2)$$

$$\mathcal{A} = \dots abc \dots abc \dots, \quad \mathcal{B} = \dots bc \dots abc \dots ab \dots \quad (1.3)$$

We observe several things. If we want to keep all occurrences of ab and bc in case (1.1) and it is already given how to match the ab duos (bc , resp.), then it is uniquely given how to match the bc duos (ab , resp.). If we want to keep all occurrences of ab and bc in case (1.2), then there is only one way how to match them all; we say that the duos have a *perfect match* and we call these matches the *perfect matches* of ab and bc . Concerning case (1.3), in any common partition of \mathcal{A} and \mathcal{B} at least one occurrence of the duos ab and bc must be broken.

Let ab be an ambiguous duo and bc a unique duo. There are two possibilities how they may interfere:

$$\mathcal{A} = \dots abc \dots ab \dots, \mathcal{B} = \dots abc \dots ab \dots \quad (2.1)$$

$$\mathcal{A} = \dots ab \dots abc \dots, \mathcal{B} = \dots ab \dots ab \dots bc \dots \quad (2.2)$$

Similarly as before, there is only one way how to match all duos in case (2.1). We say that the duos have a *perfect match* and we call these matches the *perfect matches* of ab and bc . Concerning (2.2), in any common partition of \mathcal{A} and \mathcal{B} at least one of ab and bc must be broken.

Finally, let ab and bc be two unique duos. There is only one way how they may interfere.

$$\mathcal{A} = \dots abc \dots, \mathcal{B} = \dots ab \dots bc \dots \quad (3.1)$$

Again, in any common partition of \mathcal{A} and \mathcal{B} , ab or bc must be a break.

In the second phase of the algorithm, we cut all occurrences of duos ab and bc that have an interference of type (1.3), (2.2) or (3.1). We charge these cuts to the breaks that appear, by the above observations, in the optimal solution. At most four cuts are charged to a single break.

Let \mathcal{A}_2 and \mathcal{B}_2 denote the two sets of strings after performing all cuts of phase two, and let G_2 be the corresponding conflict graph. Note that a duo might have more than one preferred match. The problem we are facing now is to decide which preferences to obey and which not since the more preferences we obey the less breaks we need.

By definition, a duo ab preference has interferences of type (1.1) only, and therefore interferes with at most two other duos. We already observed that if a duo ab has an interference of type (1.1), say with a duo bc , two matches of the duo bc are already fixed, no new breaks are allowed, then the matches of the duo ab are uniquely given. In this way, a duo without preference transmits a fixed preference of a neighboring duo on one side to a neighboring duo on its other side, etc. A difficulty arises when a preference of one duo, say bc , is transmitted by a sequence of duos without preferences to another duo with a preference, say xy , and the preference transmitted to xy is different from the preference of xy . Then, in every common partition, at least on specific duo bc , or xy , or one of the transmitting duos, must be a break. We say that the preferences of bc and xy are inconsistent. Similarly, if bc has two different preferences, we also say that bc is inconsistent with bc .

We define the graph $H = (V_H, E_H)$ of duos with inconsistent preferences. The vertex set V_H consists of all duos with a preference and the set of edges E_H consists of all pairs of duos with inconsistent preferences (which includes loops for duos inconsistent by themselves). By the above discussion, the graph H can be constructed in time linear in the number of duos.

Lemma 3. *Let H be a graph with vertex set V_H and edge set E_H . Let \mathcal{A}_2 and \mathcal{B}_2 be two sets of substrings.*

In phase three, the algorithm cuts all duos corresponding to a minimum vertex cover of H (resp., to 2-approximation of a minimum vertex cover). And we charge these cuts to the breaks in the minimum vertex cover, by the above Lemma. Let \mathcal{A}_3 and \mathcal{B}_3 denote the two sets of substrings we are left with, and let G_3 be the corresponding conflict graph.

Lemma 4. *Let \mathcal{A}_3 and \mathcal{B}_3 be two sets of substrings. Let A and B be two sets of substrings.*

We are going to construct a maximum independent set in G_3 , corresponding to a common partition of \mathcal{A}_3 and \mathcal{B}_3 with no additional breaks. We take to our independent set

- all singles, and,
- from every duo with preference, the two vertices corresponding to the preferred matches, and,
- from every duo without preference the two vertices corresponding to the matches that are forced by a neighboring duo.

In this way, every single from G_3 appears in the independent set, and for every duo with four possible matches, two of them are in the independent set. This is a maximum independent set corresponding to a common partition with no additional breaks. □

Since the cuts of the algorithm are charged in every phase to different breaks in the optimal solution, at most $4m$ breaks were used in all three phases, resulting in a 4-approximation.

Using a hash table, Phases 1 and 2 can be implemented in time $O(n)$. We already noted that the graph H can be constructed in linear time. Since the number of edges in H is $O(n)$, a 2-approximation of the vertex cover can be computed in time $O(n)$, yielding a total time $O(n)$.

For signed MCSP, the algorithm goes along the same lines, it only has to consider $+a + b$ and $-b - a$ as the occurrences of the same duo.

Theorem 5. *Sorting by reversals is NP-hard for $k \geq 4$ and for signed permutations. For $k = 3$ MCSP is NP-hard.*

Acknowledgment. We thank Xin Chen for introducing us the MCSP; this motivated our work. We wish to thank Tao Jiang, Marek Chrobak, Neal Young and Stefano Lonardi for many useful discussions. We also gratefully acknowledge comments given to us by Jiří Sgall on early version of the paper.

References

1. A. Avidor and U. Zwick. Approximating MIN k -SAT. In *Proc. of 13th International Symposium on Algorithms and Computation (ISAAC)*, volume 2518 of *Lecture Notes in Computer Science*, pages 465–475, 2002.
2. P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proc. of the 26th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 200–209, 1999.
3. A. Caprara. Sorting by reversals is difficult. In *Proc. of the First International Conference on Computational Molecular Biology*, pages 75–83, 1997.
4. X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. Submitted, 2004.
5. D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.
6. M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. In *Proc. of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2004.
7. G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proc. of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA)*, pages 667–676, 2002.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1978.
9. S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, Jan. 1999.
10. D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T. Jiang, Y. Xu, and M. Q. Zhang, editors, *Current Topics in Computational Molecular Biology*. The MIT Press, 2002.
11. D. Shapira and J. A. Storer. Edit distance with move operations. In *13th Symposium on Combinatorial Pattern Matching (CPM)*, 2002.
12. G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982.

On the Complexity of Network Synchronization

Darin Goldstein¹ and Kojiro Kobayashi²

¹ Department of Computer Engineering and Computer Science,
California State University, Long Beach
daring@cecs.csulb.edu

² Department of Information Systems Science,
Soka University
kobayasi@t.soka.ac.jp

Abstract. We show that if a minimal-time solution to a fundamental distributed computation primitive, synchronizing a network path of finite-state processors, exists on the three-dimensional, undirected grid, then we can conclude the purely complexity-theoretic result $P = NP$.

Every previous result on network synchronization for various network topologies either demonstrates the existence of fast synchronization solutions or proves that a synchronization solution cannot exist at all. To date, it is unknown whether there is a network topology for which there exists a synchronization solution but for which no *minimal-time* synchronization solution exists. Under the assumption that $P \neq NP$, this paper solves this longstanding open problem in the affirmative.

1 Introduction

The Firing Squad Synchronization Problem (or FSSP, for short) is a famous problem originally posed almost half a century ago. A prisoner is about to be executed by firing squad. The firing squad is made up of soldiers who have formed up in a straight line with muskets aimed at the prisoner. The general stands on the left side of the line, ready to give the order, but he knows that he can only communicate with the soldier to his right. In fact, each soldier can only communicate with the soldier to his immediate left and/or right, but nobody else. Soldiers have limited memory and can only pass along simple instructions. Is it possible to come up with a protocol, independent of the size of the line, for getting all of the soldiers to fire at the prisoner simultaneously if their only means of communication are small, whispered instructions only to adjacent soldiers? (The possibility of counting the number of soldiers in the line can be discounted because no soldier can remember such a potentially large amount of information; each soldier can only remember messages that are independent of the size of the line.)

The problem itself is interesting as a mathematical puzzle. More importantly, there are also applications to the synchronization of small, fast processors in large networks. In the literature on the subject (e.g. [18, 12]), the problem has been referred to as “macrosynchronization given microsynchronization” and “realizing

global synchronization using only local information exchange.” The synchronization of multiple small but fast processors in general networks is a fundamental problem of parallel processing and a computing primitive of distributed computation.

The FSSP has a rich history; solutions to various subproblems have been discovered over a period of decades. J. Mazoyer provides an overview of the problem (up to 1986, at least) in addition to some of its history in [15]. We summarize the history here. J. Myhill introduced the problem in 1957, though the first published reference is [17] from 1962. J. McCarthy and M. Minsky first solved the problem for the bidirectional line (as described above) in [16]. Later, the problem was considered for directed networks of automata. Honda and Nishitani[7] and Kobayashi[10] solved the FSSP for specific subtypes of networks, the directed ring— that is, a firing squad in the shape of a circle with the caveat that soldiers may only listen to the man on their left and speak to the man on their right— and the “ring-of-trees” — that is, networks which include a loop, containing the root, whose length is at least as great as the maximum distance from the root to any processor. Even, Litman, and Winkler[3] used this, and their invention of network-traversing “snakes” to create a protocol that would fire any strongly-connected directed network in $O(N^2)$ time. Ostrovsky and Wilkerson[18] were able to improve this to $O(ND)$ (where D refers to the diameter of the network) in 1995, which remains the best today.

Results in the area of network synchronization tend to mainly focus on introducing solutions for various network topologies and improvements in running time for these solutions. For example, after an exponential-time algorithm for firing the arbitrary directed network was discovered, Even, Litman, and Winkler in [3] were the first to show that a polynomial-time solution exists for the general directed network; Ostrovsky and Wilkerson [18] then proposed an improvement which reduced that running time. To the author’s knowledge, there are only a couple of “lower bound” results in the literature. The first is an impossibility result for the specific topology in which processors may have unbounded fan-out from any given out-port (proved in [11]); there cannot exist any solution at all for this particular topology. Additionally, a result by T. Jiang in [9, 8] indicates that there can be no solution for the case where there are multiple generals (root nodes) in the network. It has been a longstanding open problem to determine whether there is a network topology for which a synchronization solution is known to exist but a synchronization solution provably does not exist. After many years, the contribution of this paper is to finally answer this question in the affirmative under the assumption that $P \neq NP$.

1.1 The Model

As mentioned previously, we wish to model the operation of a large network of processors whose computations are all governed synchronously by the same global clock. The model is intended to mathematically abstract a physical switching network or a very large-scale parallel processing machine. The processors are designed to be small, fast, and unable to access large memory caches. Each pro-

cessor is identical and assumed to have a fixed constant number of ports which can both send and receive a constant amount of data per clock cycle. (“Constant” quantities must be independent of the size and topology of the network.)

More formally, the problem is to construct a deterministic finite-state automaton with a transition function that satisfies certain conditions. We assume that each processor in the network is identical and initially in a special “quiescent” state, in which, at each time-step, the processor sends a “blank” character through all of its ports. A processor remains in the quiescent state until a non-blank character is received by one of its in-ports. We consider connected networks of such identical synchronous finite-state automata with vertex degree uniformly bounded by a constant. These automata are meant to model very small, fast processors. The network itself may have a specific topology (see below) but potentially unbounded size. The network is formed by connecting ports of a given automaton to those of other automata with wires. Not all ports of a given automaton need necessarily be connected to other automata. The network has a global clock, the pulses between which each processor performs its computations. Processors synchronously, within a single global clock pulse, perform the following actions in order: read in the inputs from each of their ports, process their individual state changes, and prepare and broadcast their outputs. As mentioned, our network structure is specifically designed to model the practical situation of many small and fast processors performing a synchronous distributed computation. The goal of the protocol is to cause every process in the network to enter the same special “firing” state for the first time.

A solution A for a given network topology (e.g. the bidirectional line, as above) is defined to be the instantiation of an automaton with a transition function that satisfies the firing conditions outlined above for any network size. (So, by this definition, a solution for the bidirectional line must function for a bidirectional line of **any** size.) Assuming a solution A is specified, the firing time of A on a given network will refer to the number of clock cycles it takes for this network of processors programmed with the solution A to complete the protocol and simultaneously fire. The minimum firing time of a given network (of a specified topology) will refer to the minimum over all solutions A of the firing time of the network of processors programmed with solution A . A solution A_{min} for a given network topology will be a solution such that the firing time for a network of any given size (with the given topology) programmed with the algorithm A_{min} will equal the minimum firing time of the network. Note that even though the network can be of arbitrary size, the size of the algorithm A_{min} must be fixed.

1.2 Further Background and Paper Outline

Kobayashi [12] has an outline of the current research situation of minimal-time solutions to the synchronization problem for various increasingly difficult network topologies. Currently, of Kobayashi’s ten variations (lowest to highest indicating “easiest” to “hardest”), 1 through 5 have provably minimal-time solutions, and it has been shown that no solution at all can exist for variation 10. Though

variations 6 through 9 are known to have solutions, the problem of whether **minimal-time** solutions exist are considered open problems.

The purpose of this paper is to prove the following Theorem.

Theorem 1. $P \neq NP$, \iff there does not exist a minimal-time solution for FSSP on variation $7\frac{1}{2}$.

Note that, by this Theorem, the straightforward complexity-theoretic statement $P \neq NP$ implies a result about the absence of minimal-time solutions (i.e. instantiations of automata) for synchronization, two seemingly unconnected subjects. Or, equivalently, the existence of a minimal-time solution for a distributed computation synchronization primitive implies the purely complexity-theoretic result $P = NP$.

In order to prove this theorem, we need to define the three-dimensional equivalent to a problem proposed in [12, 13], the three-dimensional path extension problem (or 3-PEP), which we define rigorously below in Section 3. The proof proceeds in steps as follows.

1. We show that a minimal-time solution for FSSP on variation $7\frac{1}{2}$ implies that there exists a deterministic Turing machine that can solve the problem 3-PEP in polynomial time.
2. We prove that if $3-PEP \in P$, then two simpler versions of the 3-PEP problem, 1CUBE and 2CUBE, are also solvable in polynomial time.
3. Given the result in Step 2, we show that $3-PEP \in P \rightarrow HAM \in P$ where HAM represents a restricted NP-complete version of the Hamilton Path Problem.

Aside from being theoretically interesting, this result is important to the field of parallel processing, as fast network processor synchronization is a fundamental computing primitive. Indeed, decades of research have focused on extending the network topologies for which there exist solutions [3, 7, 10, 18], minimizing the time- and space-complexity [6, 14, 19, 20], and even proving the asymptotic time-equivalence of a number of other common network problems to FSSP [5].

The rest of the paper is organized as follows. In Section 2, we rigorously define our new variation $7\frac{1}{2}$. In Section 3, we outline the proofs of Steps 1, 2, and 3.

2 Variation $7\frac{1}{2}$

In this section, we will introduce the new network topology that will interest us for the rest of the paper. We will refer to this as variation $7\frac{1}{2}$ because it is slightly more difficult than Kobayashi's variation 7 but not quite as general as variation 8.

We can place a processor on any point $(x_1, x_2, x_3) \in \mathcal{Z}^3$ in the three-dimensional grid. We say that two processors in positions (x_1, x_2, x_3) and

(y_1, y_2, y_3) are adjacent if and only if $|x_i - y_i| = 1$ for exactly one value of i and $\forall j \neq i, x_j = y_j$. Note that processors have 6 available ports for adjacent processors: North, South, East, West, Up, and Down.

We can abstract this type of network by a three-dimensional orthogonal grid graph in which certain vertices are marked. Each marked vertex corresponds to the position of a processor, and each edge correspond to a bidirectional link between processor ports. Mathematically, the processors are abstracted by identical finite-state automata. At the outset, each automaton except a single, unique automaton, the beginning of the path, is in the quiescent state Q . Q is a special “sleeping” state: an automaton in the state Q does not send any messages and can become non-quiescent only if it receives a message from one of its ports. The root is the automaton that is initially non-quiescent. Its purpose is to begin the algorithm. (Intuitively, the root has the job of the “general.”)

Definition 1 (Variation 7 $\frac{1}{2}$): Let \mathcal{P} be a path in \mathbb{Z}^3 starting at p_1 , $p_2, \dots, p_n \in \mathbb{Z}^3$ such that p_i and p_{i+1} are adjacent for $1 \leq i < n$, p_i is not adjacent to p_j for $|i - j| > 1$. We say that \mathcal{P} is a *self-avoiding path* if and only if $p_i \neq p_j$ for $i \neq j$. We say that \mathcal{P} is a *minimal-time self-avoiding path* if and only if there is no self-avoiding path from p_1 to p_n that is shorter than \mathcal{P} .

This definition corresponds to the intuitive definition of a self-avoiding network path in three-dimensions. A solution to this variation of the FSSP trivially exists because solutions have been found for arbitrary undirected (and directed) networks. The same solutions will function on this sub-variation, just not in minimal-time.

3 The Results

In this section, we will outline Steps 1, 2, and 3 of the proof of Theorem 1.

3.1 A Minimal-Time Solution to Variation 7 $\frac{1}{2}$ Implies 3-PEP $\in P$

This section will be devoted to Step 1 of the proof of Theorem 1. First, we need to define the problem 3-PEP.

Definition 2 (3-PEP): Let \mathcal{P} be a minimal-time self-avoiding path in \mathbb{Z}^3 starting at $p_1 = (0, 0, 0)$ and ending at p_n . We say that \mathcal{P} is a *3-PEP* if and only if there is no self-avoiding path from p_1 to p_n that is shorter than \mathcal{P} .

This is precisely the three-dimensional analogue of the two-dimensional path problem presented in [12, 13].

We also define the problems 1CUBE and 2CUBE though they will not be used until Section 3.2.

Definition 3 (1CUBE): Given a graph $G = (V, E)$ with $|V| = n$ and a set of nodes $S \subseteq V$ with $|S| = k$, where $k \leq n$, and a set of nodes $T \subseteq V$ with $|T| = k$, where $k \leq n$, and a set of nodes $Z \subseteq V$ with $|Z| = k$, where $k \leq n$, and a set of nodes $U \subseteq V$ with $|U| = k$, where $k \leq n$, and a set of nodes $W \subseteq V$ with $|W| = k$, where $k \leq n$, and a set of nodes $X \subseteq V$ with $|X| = k$, where $k \leq n$, and a set of nodes $Y \subseteq V$ with $|Y| = k$, where $k \leq n$, and a set of nodes $Z^3 \subseteq V$ with $|Z^3| = k^3$, where $k \leq n$, and a set of nodes $K \subseteq V$ with $|K| = k$, where $k \leq n$, and a set of nodes $1\text{CUBE} \subseteq V$ with $|1\text{CUBE}| = k$, where $k \leq n$, and a set of nodes (n) with $|n| = n$, where $n \leq n$.

Definition 4 (2CUBE): Given a graph $G = (V, E)$ with $|V| = n$ and a set of nodes $S \subseteq V$ with $|S| = k$, where $k \leq n$, and a set of nodes $T \subseteq V$ with $|T| = k$, where $k \leq n$, and a set of nodes $Z \subseteq V$ with $|Z| = k$, where $k \leq n$, and a set of nodes $U \subseteq V$ with $|U| = k$, where $k \leq n$, and a set of nodes $W \subseteq V$ with $|W| = k$, where $k \leq n$, and a set of nodes $X \subseteq V$ with $|X| = k$, where $k \leq n$, and a set of nodes $Y \subseteq V$ with $|Y| = k$, where $k \leq n$, and a set of nodes $Z^3 \subseteq V$ with $|Z^3| = k^3$, where $k \leq n$, and a set of nodes $K \subseteq V$ with $|K| = k$, where $k \leq n$, and a set of nodes $2\text{CUBE} \subseteq V$ with $|2\text{CUBE}| = k$, where $k \leq n$, and a set of nodes (n) with $|n| = n$, where $n \leq n$.

The proof that a minimal-time FSSP solution to variation $7\frac{1}{2}$ implies $3-PEP \in P$ is a direct three-dimensional adaptation of that given in [12, 13]. Due to space constraints, we are unable to present a complete proof. For a thorough treatment, we recommend consulting the stated references.

Theorem 2. ($\text{3-PEP} \in P \rightarrow \text{1CUBE} \in P$ and $\text{3-PEP} \in P \rightarrow \text{2CUBE} \in P$).
 Given a graph $G = (V, E)$ with $|V| = n$ and a set of nodes $S \subseteq V$ with $|S| = k$, where $k \leq n$, and a set of nodes $T \subseteq V$ with $|T| = k$, where $k \leq n$, and a set of nodes $Z \subseteq V$ with $|Z| = k$, where $k \leq n$, and a set of nodes $U \subseteq V$ with $|U| = k$, where $k \leq n$, and a set of nodes $W \subseteq V$ with $|W| = k$, where $k \leq n$, and a set of nodes $X \subseteq V$ with $|X| = k$, where $k \leq n$, and a set of nodes $Y \subseteq V$ with $|Y| = k$, where $k \leq n$, and a set of nodes $Z^3 \subseteq V$ with $|Z^3| = k^3$, where $k \leq n$, and a set of nodes $K \subseteq V$ with $|K| = k$, where $k \leq n$, and a set of nodes $7\frac{1}{2}$ with $|7\frac{1}{2}| = k$, where $k \leq n$, and a set of nodes $3-PEP \in P$ with $|3-PEP \in P| = k$, where $k \leq n$.

3.2 $3-PEP \in P \rightarrow 1CUBE \in P$ and $3-PEP \in P \rightarrow 2CUBE \in P$

In this section, we complete Step 2 of Theorem 1.

Before proceeding to the statement of Theorem 3, we need to take note of the following fact. Paths can, in many cases, carve out “solid” structures; we recommend glancing at Figure 1 for the rough definitions of the basic building blocks we will use.

Theorem 3. $3-PEP \in P \rightarrow 1CUBE \in P$ and $3-PEP \in P \rightarrow 2CUBE \in P$

Due to space constraints, we omit the proof of this theorem. Note that because the 1CUBE and 2CUBE decision problems are solvable in polynomial time, it

¹ The reason we represent the number n in unary here and below in Definition 4 is that we wish to be able to construct the cube in Z^3 as a polynomial-time part of the problem instance. Note that in Definition 2, the path is also explicitly constructed.
² The term *avoiding* refers to the fact that the path must not only avoid itself but the walls of the cube as well. The path is not allowed to become adjacent to any wall. We assume that the opening in the cube is just large enough to accept a single path entry that manages to avoid the walls, an opening of edge length 4×4 . We use this term in the same manner in Definition 4.

is trivial to construct a polynomial time algorithm to determine the length of the longest path that enters an $n \times n \times n$ cube once through the middle of a face and remains inside and a polynomial time algorithm for determining the length of the longest path completely enclosed by the $n \times n \times n$ cube that is required to begin at the “entrance” and end at the “exit”.

3.3 3 – PEP $\in P \rightarrow HAM \in P$

In this section, we will justify Step 3 in the proof of Theorem 1.

In order to do so, we need to introduce an operation on graphs in \mathcal{Z}^3 called a blow-up by a factor of k . The idea behind this operation is simple. We apply the linear map $(x_1, x_2, x_3) \mapsto (kx_1, kx_2, kx_3)$ to every point in \mathcal{Z}^3 . Note that the blow-up operation preserves slopes of lines, but increases distances by a factor of k .

Theorem 4 (Step 3 in the proof of Theorem 1). *3 – PEP $\in P \rightarrow HAM \in P$*

Assume that we are given a planar, cubic graph G and that $3 - PEP \in P$. We first choose two vertices in G , v_{start} and v_{end} , and remove two edges from each vertex, leaving only one edge remaining per vertex. These vertices will correspond to the start and end of a potential Hamilton Path in the graph G . Call the new graph G' .

Our first nontrivial goal will be to illustrate how it is possible to build a three-dimensional connected structure out of a single path that has the same connectivity properties as G' . The structure will contain an “inside” and an “outside” such that a path originating on the inside of the structure cannot possibly make it to the outside and vice versa. The beginning of the path (root) will be required to be outside the structure and the end of the path will be inside the structure. We will then show how it is possible to solve the Hamilton Path Problem on G utilizing a solution to the 3 – PEP problem.

A integer-grid straight-line drawing of a planar graph G is an embedding of G in the two-dimensional integer grid such that the vertices of G are grid points and the edges of G are non-crossing straight lines. In [2, 1], it is shown how an n -vertex planar graph can be embedded as an integer grid straight-line drawing into the plane in $O(n^2)$ space and polynomial time. Embed the graph G' in the xy plane (i.e. $z = 0$) in \mathcal{Z}^3 in this way.

In the following discussion, we will assume that n is large enough to dwarf all constants hidden by the asymptotic notation.

We now have a planar graph in \mathcal{Z}^3 with no crossing edges. Blow-up \mathcal{Z}^3 by a factor of n^5 . Note that, by elementary geometry, any vertex or line without an endpoint at v must have been at least distance $\Omega(\frac{1}{n})$ from v before the blow-up. Blow-ups increase distances by factors. Thus, for any vertex v , we can be

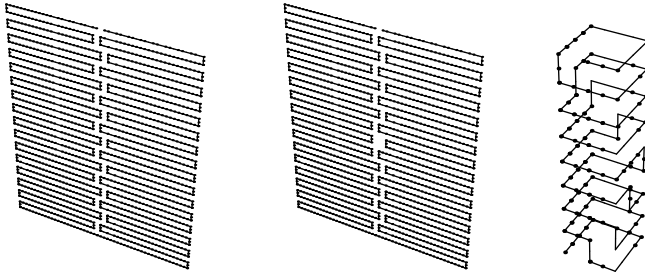


Fig. 1. This figure illustrates some of the simple building blocks we will use to create the more complicated structures below. The leftmost picture is a closed wall formed by a single path. Note that even though there is not a solid mass of vertices, a continuation of the path could not possibly pass through the wall, as it would violate the definition of a path (Definition 1). In the center is an “open” wall. Note that because the height of the hole in the center is 6 edges tall and 4 edges wide, it is possible for the path to pass through the center of this structure without violating any path properties. Finally, the rightmost picture is an illustration of a “one-way street.” Note that, if necessary, the path could potentially pass through the dead center of the 4 edge by 4 edge cross-section of the passage without being adjacent to any other processors. However, the path could only pass through once (i.e. in a single direction)

guaranteed that all other vertices and lines without an endpoint at v are at least a distance $\Omega(n^4)$ from v . Around each vertex v , we can draw the following figure: (a) an “inner” 3D cube with v at the center, two 4×4 openings on adjacent sides, and with edge size $\Theta(n^3)$ and (b) a “surrounding” 3D cube with v at the center, three 4×4 openings on adjacent sides (i.e. all sides having an opening meet at a single vertex), and with edge size $\Theta(1)$ larger than the first cube. (The outer cubes of v_{start} and v_{end} will only have one opening each.) To form each cube, we make use of our building blocks from Figure 1. The formation of a small surrounding cube can be seen in Figure 3. We are guaranteed that no surrounding cube comes within a distance of $\Omega(n^4)$ of any other surrounding cube. Once the cubes have been drawn, delete the vertex v . (In other words, we are replacing vertices with cubes.)

At this point, we have a three-dimensional situation in which the vertices are represented by two large cubes, one inside the other, and are connected by straight line segments. We now concentrate on replacing the edges of the original straight-line drawing with three-dimensional structures. Our goal is to replace the straight lines that would otherwise enter the vertices with three-dimensional paths. See Figure 2. Of course, these straight line segments do not necessarily proceed along a rectilinear path, as the three-dimensional one-way paths do, and we are therefore required to use the third dimension: If a one-way path needs to make a rectilinear turn, we simply direct it into a small $\Theta(1)$ -sized

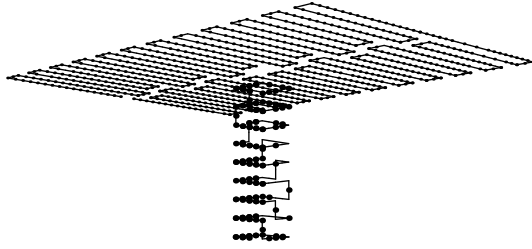


Fig. 2. This figure illustrates the connection of an open cube face via a one-way street, one of the building blocks in Figure 1. Note that the one-way street stops at a distance of 2 units away from the face of the cube. If there were a cube connector passing down the front of the cube face, it would have to make a detour around the one-way street so as not to interfere. Placing a one-way street at the entrance of a cube forces the 6 by 4 entrance to also be one-way

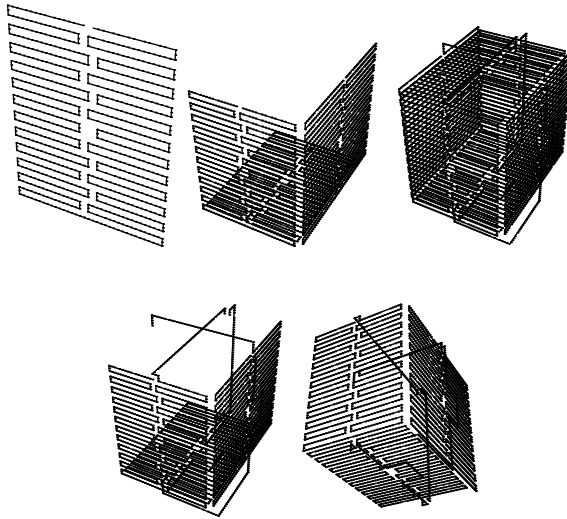


Fig. 3. This figure illustrates the formation of a cube using two of the building blocks (closed and open walls) from Figure 1. Notice the opening in the front face (made from the open wall, first picture). We can make an opening in any face of the cube in this way. We have made openings in the front, right, and bottom faces of the cube (second picture). The back, left, and top faces are all closed, completing the final cube (third picture). Note also that connectors are necessary between the faces; this entire structure is made from a *single* path. The path begins at the top of the front face and ends at the top of the left side. In the bottom two pictures, we emphasize the connectors by whiting out the left, back, and top walls and displaying the resulting half-cube from different angles. Obviously, much larger cubes are possible using the same basic structures. Note that the connectors make detours around the openings in the cube on the right and bottom so that one-way streets can fit on the openings

box³ and open a second face of the box in the direction that we wish to turn, continuing a second one-way path out of this new opening. We replace every straight line segment by an appropriate string of one-way streets that turn as necessary.

In order to make the new three-dimensional structure have the same connectivity properties as G' , we need to guarantee that all one-way streets are able to enter and leave the appropriate cubes without crossing or interfering with the rest of the structure. Recall that the original graph G' was embedded in the xy plane in \mathcal{Z}^3 . Number each of the $O(n^2)$ edges of G' . Note that if we proceed upwards in the z direction, after $\Theta(n^3)$ distance (the maximum distance that any cube stretches outwards), the space is structure-free. Assign each edge a unique space in which to maneuver from cube to cube. Let the edge size of the “turn” box for the one-way streets be $c_{turn} = \Theta(1)$. Edge i will receive the grid space between $(c_{turn} + 4) * i + \Theta(n^3)$ and $(c_{turn} + 4) * (i + 1) + \Theta(n^3)$ in the z direction. (i.e. Edge i receives all grid points of the form (x, y, z) where $(c_{turn} + 4) * i + \Theta(n^3) \leq z < (c_{turn} + 4) * (i + 1) + \Theta(n^3)$.) To connect Cubes A and B with one-way streets: upon leaving Cube A , we simply make a turn directly upwards (if necessary), move to the appropriate maneuver space, proceed to the spot above Cube B using only at most 2 more turns, turn downwards towards Cube B , and then turn towards Cube B (if necessary) to enter. In this way, the new three-dimensional graph has the same connectivity properties as the original graph G' .

At this point, we have a three-dimensional structure that is broken into many separate pieces. We connect these pieces into one long path by connecting the beginning of one structure with the end of the next structure. Let the end of the path terminate at the one of the holes of the inner cube of v_{start} , forcing any extension into the cube. We now have a connected three-dimensional structure G'_{3D} that represents the graph G' in three-dimensions, where the vertices are large cubes, edges are three-dimensional one-way streets, and the entire structure is drawn with a single path originating outside the structure and terminating in such a way that any extension must continue inside the structure.

We now examine the structure G'_{3D} . Because $3-PEP \in P$ by assumption, we know from Theorem 3 that $1CUBE \in P$ and $2CUBE \in P$. By the subsequent remarks, we can determine the maximum path length that a given cube of either type admits in polynomial time in n . Let this maximum path length be $L_1(n)$ for a one-holed inner cube and $L_2(n)$ for a two-holed inner cube. Both quantities are clearly $\Theta(n^9)$.

Because the path originates outside the structure G'_{3D} we have formed, we can extend the beginning (the root) out as far as we like. Extend the beginning of the path so that the length of the path structure G'_{3D} ends up being $(n -$

³ Terminology: Note that a “box” is used for a turn and are always constant-sized. “Cubes” are the structures that replaced the vertices above and all have edge size $\Theta(n^3)$. We use to different words to distinguish between these two even though both shapes are the same; their size and utility are different.

1) $L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$. We then ask the following question (3-PEP): Can we extend the path given by G'_{3D} to double its length? (i.e. Can we extend G'_{3D} by length at least $(n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$ and still have the resulting structure satisfy the properties for a path in Definition 1?)

We claim that there exists two vertices v_{start} and v_{end} and two removed edges per vertex such that the answer to this question for G'_{3D} is YES if and only if a Hamilton path exists in the graph G .

There are two cases to consider.

1. Assume that there is a Hamilton path in the graph G that starts at v_{start} , uses the edge from v_{start} that was not removed, and terminates at v_{end} via the single edge not removed. Then G'_{3D} has a path extension that enters and exits exactly $n - 1$ inner cubes and enters and remain inside one additional inner cube. The length of this path extension is at least $(n - 1)L_2(n) + L_1(n)$ and is therefore greater than $(n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$.
2. Assume that there is NOT a Hamilton path in the graph G that starts at v_{start} , uses the edge from v_{start} that was not removed, and terminates at v_{end} via the single edge not removed. In this case, because the graph is cubic, any path in G' that visits every vertex must start at v_{start} , end at v_{end} , and traverse at least one edge more than once. To see this, assume the contrary. Obviously there cannot exist a vertex other than v_{start} and v_{end} that utilizes only one of its edges. If every vertex other than these two uses exactly two edges, then we have a Hamilton path. Thus, there must exist at least one vertex that uses exactly three edges. But this is also clearly impossible.

Because our construction specifically forbids this, a maximal path extension cannot possibly visit every cube. Consider an extension to G'_{3D} that does not enter every inner cube in the drawing. The total length of all of the “one-way street” edges is at most $O(n^8)$ because there are at most $O(n^2)$ edges, each of which has “length” at most $O(n^6)$. If we skip at least one cube, then the most the path can be is $(n - 2)L_2(n) + L_1(n) + O(n^8) < (n - 1)L_2(n) + L_1(n) - \lfloor \frac{L_2(n)}{2} \rfloor$.

We need to check each possibility for combinations of edge removals, a polynomial number in n . If a single one gets a YES result, then the answer to *HAM* is YES. Otherwise, the answer is NO.

□

References

1. H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting straight-line embeddings of planar graph. In *Proc. of the 20th Annual ACM Symp. on Theory of Computing*, pages 426–433, 1988.
2. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
3. S. Even, A. Litman, and P. Winkler. Computing with snakes in directed networks of automata. *J. Algorithms*, 24(1):158–170, 1997.

4. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
5. D. Goldstein and N. Meyer. The wake up and report problem is asymptotically time-equivalent to the firing squad synchronization problem. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 578–587, San Francisco, CA, 6–8 January 2002.
6. E. Goto. A minimal time solution of the firing squad problem. *Course Notes for Applied Mathematics 298, Harvard University*, pages 52–59, 1962.
7. N. Honda and Y. Nishitani. The firing squad synchronization problem for graphs. *Theoretical Computer Science*, 14(1):39–61, 1981.
8. T. Jiang. The synchronization of nonuniform networks of finite automata. In *Proc. of the 30th Annual ACM Symp. on Foundations of Computer Science*, pages 376–381, 1989.
9. T. Jiang. The synchronization of nonuniform networks of finite automata. *Information and Computation*, 97:234–261, 1992.
10. K. Kobayashi. The firing squad synchronization problems for a class of polyautomata networks. *J. Comput. System Sci.*, 17(3):300–318, 1978.
11. K. Kobayashi. On the minimal firing time of the firing squad synchronization problem for polyautomata networks. *Theoretical Computer Science*, 7:149–167, 1978.
12. K. Kobayashi. A complexity-theoretical approach to the firing squad synchronization problem. In *Proceedings of JIM '99 (Journée de l'Informatique Messine (Days of Metz Informatics))*, “NP-Completeness and Parallelism”, Metz University, Institute of Technology, 17–18 May 1999.
13. K. Kobayashi. On time optimal solutions of the firing squad synchronization problem for two-dimensional paths. *Theoretical Computer Science*, 259:129–143, 28 May 2001.
14. J. Mazoyer. A six states minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50:183–238, 1987.
15. J. Mazoyer. An overview of the firing synchronization problem. In *Automata Networks, LITP Spring School on Theoretical Computer Science, Angèlès-Village, France, May 12-16, 1986, Proceedings*, volume 316 of *Lecture Notes in Computer Science*. Springer, 1988.
16. M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1967.
17. E. F. Moore. *Sequential Machines, Selected Papers*. Addison Wesley, Reading, MA, 1962.
18. R. Ostrovsky and D. Wilkerson. Faster computation on directed networks of automata. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 38–46, Ottawa, Ontario, Canada, 2–23 August 1995.
19. A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9:66–78, 1966.
20. J.-B. Yunes. Seven states solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 127(2):313–332, 1993.

Counting Spanning Trees and Other Structures in Non-constant-jump Circulant Graphs* (Extended Abstract)

Mordecai J. Golin, Yiu Cho Leung, and Yajun Wang

Department of Computer Science, HKUST, Clear Water Bay,
Kowloon, Hong Kong
{golin, cho, yalding}@cs.ust.hk

Abstract. Circulant graphs are an extremely well-studied subclass of regular graphs, partially because they model many practical computer network topologies. It has long been known that the number of spanning trees in n -node circulant graphs with *constant jumps* satisfies a recurrence relation in n . For the non-constant-jump case, i.e., where some jump sizes can be functions of the graph size, only a few special cases such as the Möbius ladder had been studied but no general results were known.

In this note we show how that the number of spanning trees for *all* classes of n node circulant graphs satisfies a recurrence relation in n even when the jumps are non-constant (but linear) in the graph size. The technique developed is very general and can be used to show that many other structures of these circulant graphs, e.g., number of Hamiltonian Cycles, Eulerian Cycles, Eulerian Orientations, etc., also satisfy recurrence relations.

The technique presented for *deriving* the recurrence relations is very mechanical and, for circulant graphs with small jump parameters, can easily be quickly implemented on a computer. We illustrate this by deriving recurrence relations counting all of the structures listed above for various circulant graphs.

1 Introduction

The purpose of this note is to develop techniques for counting structures, e.g., Spanning Trees, Hamiltonian Cycles, Eulerian Cycles, Eulerian Orientations, Matchings, etc., in circulant graphs with non-constant jumps. We start off by defining circulant graphs and reviewing the large literature on counting spanning trees in constant-jump circulant graphs and then the much lesser literature on counting spanning trees in non-constant-jump ones.

* The work of the first two authors was partially supported by HK CERG grants HKUST6162/00E, HKUST6082/01E and HKUST6206/02E. The third author was partially supported by HK CERG grant HKUST 6190/02E.

Definition 1. The n -vertex undirected circulant graph $C_n^{s_1, s_2, \dots, s_k}$ has vertex set $\{0, 1, 2, \dots, n-1\}$ and edge set $E_C(n) = \{(i, j) : i - j \pmod n \in \{s_1, s_2, \dots, s_k\}\}$.

$$V(n) = \{0, 1, \dots, n-1\} \quad E_C(n) = \{(i, j) : i - j \pmod n \in \{s_1, s_2, \dots, s_k\}\}.$$

The simplest circulant graph is the n vertex cycle C_n^1 . The next simplest is the $C_n^{1,2}$ in which every vertex is connected to its two neighbors and neighbor's neighbors. The lefthand sides of figures 1, 2 and 3 illustrate various circulant graphs. Circulant graphs (sometimes known as "loop networks") are very well studied structures, in part because they model practical data connection networks [12, 3].

One frequently studied parameter of circulant graphs is the number of spanning trees they have. For connected graph G , $T(G)$ denotes the number of spanning trees in G . $T(G)$ is examined both for its own sake and because it has practical implications for network reliability, e.g., [8, 9]. For any graph G , Kirchoff's theorem [13] efficiently permits calculating $T(G)$ by evaluating a co-factor of the Laplacian matrix of G (this essentially calculates the determinant of a matrix related to the adjacency matrix of G .) The interesting problem, then, is not in calculating the number of spanning trees in a graph, but in calculating the number of spanning trees as a function of a parameter, in graphs chosen from a family defined by a parameter.

The first result in this area for circulant graphs was the fact that $T(C_n^{1,2}) = nF_n^2$, F_n the Fibonacci numbers, i.e., $F_n = F_{n-1} + F_{n-2}$ with $F_1 = F_2 = 1$. This result, originally conjectured by Bedrosian [2] was subsequently proven by Kleitman and Golden [14]. (The same formula was also conjectured by Boesch and Wang [5] without knowledge of [14].)

Later proofs of $T(C_n^{1,2}) = nF_n^2$, and analyses of $T(C_n^{s_1, s_2, \dots, s_k})$ as a function of n for special values of s_1, s_2, \dots, s_k can be found in [1, 21, 6, 24, 19, 23]. A general result due to [21] and later [25] is that, for any $1 \leq s_1 < s_2 < \dots < s_k$, $T(C_n^{s_1, s_2, \dots, s_k})$ satisfies a constant coefficient linear recurrence relation of order $2^{2s_k-2} - 1$.

Knowing the form of the recurrence relation permits explicitly constructing it by using Kirchoff's theorem to evaluate $T(C_n^{s_1, s_2, \dots, s_k})$ for enough values of n to solve for the coefficients of the recurrence relation.

With the exception of [14], which was a combinatorial proof using techniques very specific to the $C_n^{1,2}$ case, all of the results above were algebraic. That is they all worked by evaluating the co-factor of the Kirchoff matrix of the graphs. These co-factors could be expressed in terms of the eigenvalues of the adjacency matrices of the circulant graphs and the eigenvalues of these adjacency matrices (also known as circulant eigenvalues) are well known [4]. All of the results mentioned took advantage of these facts and essentially consisted of clever algebraic manipulations of these known terms. The recurrence relations for $T(C_n^{s_1, s_2, \dots, s_k})$

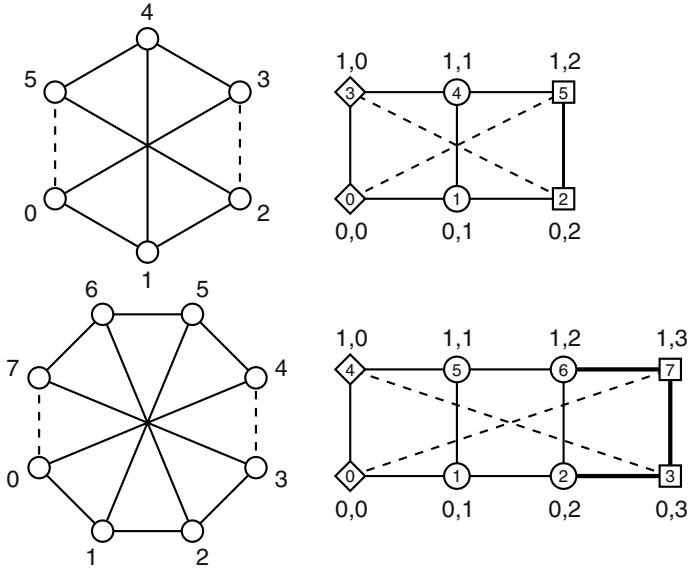


Fig. 1. The Möbius ladder $C_{2n}^{1,n}$ for $n = 3, 4$ drawn in both circulant form and lattice form. The solid edges in the figures on the right are $L_n^{1,2}$. The dashed edges are $E_C(n) - E_L(n)$. The bold edges are $E_L(n) - E_L(n - 1)$. The diamond vertices are $L(n)$ while the square vertices are $R(n)$

popped out of these manipulations but did not possess any explicit combinatorial meaning.

In a recent paper [11] two of the authors of this note introduced a new technique, \dots , for counting spanning trees in circulant graphs with \dots jumps. This technique was purely combinatorial and therefore permitted a combinatorial derivation of the recurrence relations on $T(C_n^{s_1, s_2, \dots, s_k})$. It also permitted deriving recurrence relations for the number of Hamiltonian Cycles, Eulerian Cycles, Eulerian Orientations and other structures in circulant graphs with constant jumps.

An open question in [11] was whether there was \dots general technique, combinatorial or otherwise, for counting structures in circulant graphs with non-constant jumps, i.e., graphs in which the jumps are a function of the graph size. The canonical example of such graphs is the Möbius ladder $C_{2n}^{1,n}$. (See Figure 1; Two other non-constant-jump circulant graphs, $C_{3n}^{1,n}$ and $C_{3n+1}^{1,n}$, are illustrated in Figures 2 and 3.) It is well known that

$$T(C_{2n}^{1,n}) = \frac{n}{2} \left[(2 + \sqrt{3})^n + (2 - \sqrt{3})^n + 2 \right] \tag{1}$$

According to [6] this result is due to [18]. Other proofs can be found in [15, 17] (combinatorial) and [6] (algebraic). The combinatorial proofs are very specially crafted for the Möbius ladder and do not seem generalizable to other circulant

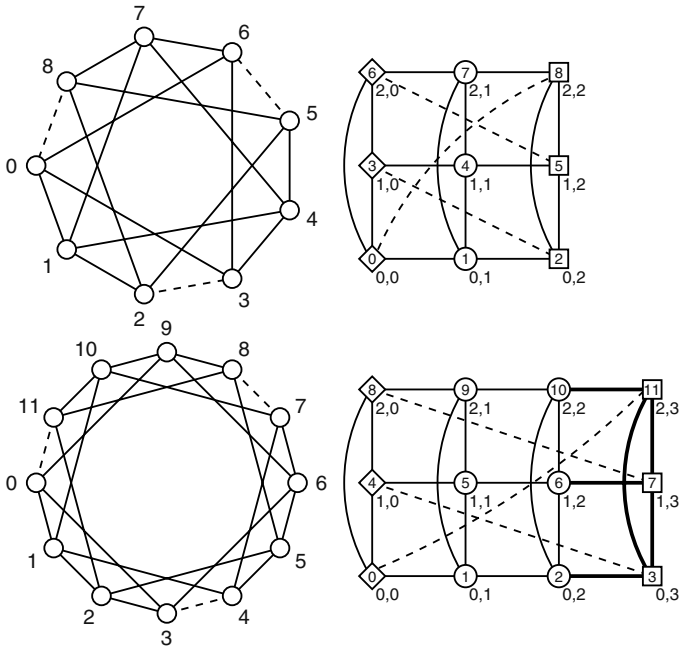


Fig. 2. $C_{3n}^{1,n}$ and $L_{3n}^{1,n}$ for $n = 3, 4$

graphs. The technique of [6] again consisted of using algebraic techniques, this time clever manipulation of Chebyshev polynomials, to evaluate a co-factor of the Kirchoff-Matrix. [10] showed how to push this Chebyshev polynomial technique slightly further to count the number of spanning trees in a small number of very special non-constant-jump circulant graphs.

The major result of this paper is a technique for counting structures in circulant graphs with linear jumps. More specifically,

Theorem 1.

$\mathcal{A} \in \{\text{Spanning Trees, Hamiltonian Cycles, Eulerian Cycles, Eulerian Orientations}\}$

Let $G = C_{pn+s}^{p_1, p_2, \dots, p_k, s_1, s_2, \dots, s_k}$ be a circulant graph with $pn+s$ nodes and k linear jumps p_1, p_2, \dots, p_k and s_1, s_2, \dots, s_k constant jumps, where $\forall i, p_i < p$.

$$C_n = C_{pn+s}^{p_1 n + s_1, p_2 n + s_2, \dots, p_k n + s_k}$$

$$T^{\mathcal{A}}(n) = |\mathcal{A}(C_n)|$$

$$T^{\mathcal{A}}(n) \text{ satisfies a recurrence relation of order } n$$

Note that if \mathcal{A} is Spanning Trees, $p = 1, s = 0$ and $\forall i, p_i = 0$ then this collapses to the known fact [21, 25] that the number of spanning trees in circulant graphs with constant jumps satisfies a recurrence relation.

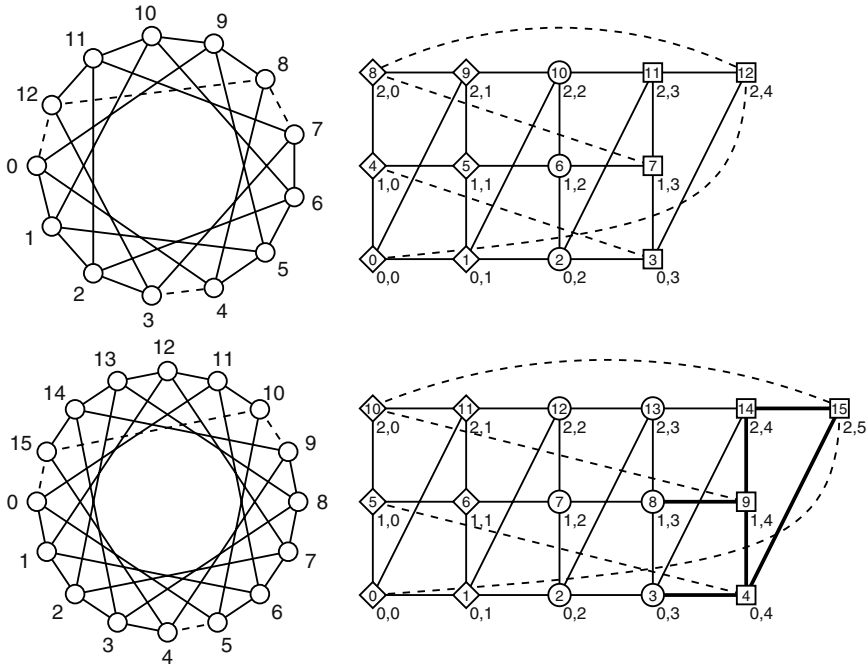


Fig. 3. $C_{3n+1}^{1,n}$ and $L_{3n+1}^{1,n}$ for $n = 4, 5$

All circulant graphs are shown in circulant form and lattice form. The solid edges in the figures on the right are the corresponding lattice graphs. The dashed edges are $E_C(n) - E_L(n)$. The bold edges are $E_L(n) - E_L(n - 1)$. The diamond vertices are $L(n)$ while the square vertices are $R(n)$. Note that all edges in $E_C(n) - E_L(n)$ are in $R(n) \times L(n)$. Also note that all edges in $E_L(n + 1) - E_L(n)$ are in $(R(n + 1) - R(n)) \times (R(n) \cup R(n + 1))$.

The proof of Theorem 1 is “constructive”. That is, it shows how to build such a recurrence relation. The construction is also mechanical; that is, it is quite easy to program a computer to derive the recurrences. As examples, we have calculated the recurrence relations for various graphs which are presented in Table 1. We point out, though, that the main contribution of Theorem 1 is the \dots of such recurrence relations. The recurrence relations (and therefore the construction) can grow exponentially in p, s and the s_i and can therefore quickly become infeasible to implement.

Technically, the major contribution of this paper is a new \dots of non-constant-jump circulant graphs in terms of \dots . This new representation will permit using the machinery developed in [11] to derive recurrence relations.

In the next section we introduce lattice graphs and describe how to represent circulant graphs in terms of them. We also prove properties of lattice graphs that will enable us to use them to derive recurrence relations on the number of structures. In Section 3 we then use these properties to prove Theorem 1 when

Table 1. Some sample results derived using Theorem 1. Note that for $C_{2n}^{1,n}$, the *Möbius ladder*, $T(n)$ is equivalent to (1) and was already derived in [18, 15, 17, 6] while $H(n)$ was given in [15]. The other results in the table all new. Note that the number of Eulerian Orientations and Cycles of the Möbius ladder was not calculated. This is because, as a 3-regular graph, it does not have *any* Eulerian Orientations or Cycles

C_n	Number of structures in C_n as function of n
	Spanning trees
$C_{2n}^{1,n}$	$T(n) = 10T(n-1) - 35T(n-2) + 52T(n-3) - 35T(n-4) + 10T(n-5) - T(n-6)$ with initial values 16, 81, 392, 1815, 8112, 35301 for $n = 2, 3, 4, 5, 6, 7$
$C_{2n+1}^{1,n}$	$T(n) = 16T(n-1) - 80T(n-2) + 130T(n-3) - 80T(n-4) + 16T(n-5) - T(n-6)$ with initial values 125, 1183, 10404, 87131, 705757, 5581500 for $n = 2, 3, 4, 5, 6, 7$
$C_{3n}^{1,n}$	$T(n) = 58T(n-1) - 1131T(n-2) + 8700T(n-3) - 29493T(n-4) + 43734T(n-5) - 29493T(n-6) + 8700T(n-7) - 1131T(n-8) + 58T(n-9) - T(n-10)$ with initial values 384, 12321, 371712, 10634460, 292771602, 7840133364, 205687578624, 5312055930723, 135495271297920, 3421537009450692 for $n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11$
	Hamiltonian cycles
$C_{2n}^{1,n}$	$H(n) = H(n-1) + H(n-2) - H(n-3)$ with initial values 3, 6, 5 for $n = 2, 3, 4$ $= \begin{cases} n+1 & n \text{ even} \\ n+3 & n \text{ odd} \end{cases}$
$C_{2n+1}^{1,n}$	$H(n) = 3H(n-1) - H(n-2) - 2H(n-3) + H(n-5)$ with initial values 12, 23, 41, 79, 158 for $n = 2, 3, 4, 5, 6$
	Eulerian cycles
$C_{3n}^{1,n}$	$EC(n) = 47EC(n-1) - 742EC(n-2) + 4796EC(n-3) - 13144EC(n-4) + 12320EC(n-5)$ with initial values 372, 8924, 209228, 4798236, 108376972 for $n = 2, 3, 4, 5, 6$
	Eulerian orientations
$C_{3n}^{1,n}$	$EO(n) = 7EO(n-1) - 14EO(n-2) + 8EO(n-3)$ with initial values 38, 142, 542 for $n = 2, 3, 4$

\mathcal{A} is “Spanning Trees”. The proof is actually constructive; in the Appendix we walk through this construction to rederive the number of spanning trees in $C_{2n}^{1,n}$ as a function of n .

We finish this section by pointing out that counting the number of Hamiltonian Cycles, Eulerian Orientations and Eulerian Cycles in general undirected graphs is, in a qualitative way, very different from counting the number of spanning trees. While the Kirchoff matrix technique provides a polynomial time algorithm for counting the number of spanning trees, it is known that counting Hamiltonian Cycles [20], Eulerian Orientations [16] and Eulerian Cycles [7] in a general undirected graph is #P-complete. When the problem is restricted to circulant graphs, with the exception of [15] which counts the number of Hamiltonian Cycles in the Möbius ladder and [22], which analyzes Hamiltonian cycles in constant jump circulant graphs with $k = 2$ we know of no results (other than the previously mentioned [11]) counting these values.

2 Lattice Graphs

The major impediment to deriving recurrence relations relating the number of structures in circulant graphs to the number of structures in smaller circulant graphs is that it is difficult to see how to decompose C_n in terms of C_m where $m < n$, i.e.,

The main result for this section is a way of visualizing a circulant graph $C_n = C_{pn+s}^{p_1n+s_1, p_2n+s_2, \dots, p_kn+s_k}$ as a L_n plus a constant number of extra edges where the constant

The reason for taking this approach is that, unlike the circulant graph, the lattice graphs built recursively with L_{n+1} being L_n with the addition of a constant (independent of n) set of edges.

Before starting we will first have to rethink the way that we define circulant graphs.

Definition 2. Let $p, s, p_1, p_2, \dots, p_k, s_1, s_2, \dots, s_k$ be integers such that $pn + s = (p_1n + s_1) + (p_2n + s_2) + \dots + (p_kn + s_k)$. Then the circulant graph $C_n = C_{pn+s}^{p_1n+s_1, p_2n+s_2, \dots, p_kn+s_k}$ is defined as $C_n = L_n \cup E_C(n)$ where $E_C(n) = \{(u, v) \in E_C(n) : f(n; u, v) = un + v\}$.

$$\widehat{C}_n = (\widehat{V}_C(n), \widehat{E}_C(n))$$

where

$$\widehat{V}(n) = \{(u, v) : 0 \leq u \leq p - 1, 0 \leq v \leq n - 1\} \cup \{(p - 1, v) \mid n \leq v \leq n + s - 1\}$$

$$\widehat{E}_C(n) = \bigcup_{i=1}^k \left\{ \{(u_1, v_1), (u_2, v_2)\} : (u_1, v_1), (u_2, v_2) \in \widehat{V}(n) \text{ and } f(n; u_2, v_2) - f(n; u_1, v_1) \equiv p_i n + s_i \pmod{pn + s} \right\}$$

Directly from the definition we see \widehat{C}_n is isomorphic to $C_n = C_{pn+s}^{p_1n+s_1, p_2n+s_2, \dots, p_kn+s_k}$. Figures 1 2 and 3 illustrate this isomorphism; the graphs on the left are the circulant graphs drawn in the normal way; the graphs on the right, the new way, with node labelling showing the isomorphism.

Since the graphs are isomorphic, counting structures in C_n is equivalent to counting structures in \widehat{C}_n . Therefore, we will replace C_n by \widehat{C}_n . That is C_n will refer to \widehat{C}_n , $V(n)$ to $\widehat{V}(n)$ and $E_C(n)$ to $\widehat{E}_C(n)$. Given this new representation we can now define

Definition 3 (Lattice Graphs). Let $p, s, p_1, p_2, \dots, p_k, s_1, s_2, \dots, s_k$ be positive integers. The **Lattice Graph**

$$L_n = L_{pn+s}^{p_1n+s_1, p_2n+s_2, \dots, p_kn+s_k}$$

$$L_n = (V(n), E_L(n)) \quad V(n) = \widehat{V}(n)$$

$$E_L(n) = \bigcup_{i=1}^k \left\{ \begin{array}{l} \{(u_1, v_1), (u_2, v_2)\} : (u_1, v_1), (u_2, v_2) \in \widehat{V}(n), \\ f(n; u_2, v_2) - f(n; u_1, v_1) \equiv p_i n + s_i \pmod{pn + s}, \\ \text{and } u_2 - u_1 \equiv p_i \pmod{p} \end{array} \right\}$$

In Figures 1, 2 and 3 the solid edges in the graphs on the right form the Lattice Graphs. By comparing the definition of $E_L(n)$ in this definition to that of $E_C(n) = \widehat{E}_C(n)$ given in Definition 2 we immediately have that $E_L(n) \subseteq E_C(n)$. Referring back to the examples in Figures 1, 2 and 3 again we see that the edges in $E_C(n) - E_L(n)$, the dashed edges, always seem to connect vertices on the left (diamonds) with vertices on the right (squares). This simple observation will be at the core of our analysis. We first need the following definition:

Definition 4. Let $s_{max} = \max\{s, s_1, s_2, \dots, s_k\}$.

$$L(n) = \left\{ (u, v) : 0 \leq u \leq (p-1), 0 \leq v \leq s + s_{max} - 1 \right\}, \quad \text{Left Vertices}$$

$$R(n) = \left\{ (u, v) : 0 \leq u \leq (p-1), v \geq n - s_{max} \right\}, \quad \text{Right Vertices}$$

In order to make our analysis work we will require that $L(n) \cap R(n) = \emptyset$. To ensure this we will, from now on, require that $n > s + 2s_{max} - 1$.

Suppose $e = \{(u_1, v_1), (u_2, v_2)\} \in E_C(n)$ is a jump $p_i n + s_i$ from (u_1, v_1) . There are two cases:

1. The jump does not “cross” vertex $(p-1, n+s-1)$:

$$u_2 n + v_2 - (u_1 n + v_1) = p_i n + s_i. \tag{2}$$

If $e \in E_L(n)$, we have $u_2 - u_1 = p_i$ and $v_2 - v_1 = s_i$. If $e \in E_C(n) - E_L(n)$, because $n > s + 2s_{max} - 1$, we will have $v_2 - v_1 - s_i = (p_i - u_2 + u_1)n$, so $u_2 - u_1 = p_i \pm 1$.

2. The jump crosses vertex $(p-1, n+s-1)$:

$$u_2 n + v_2 - (u_1 n + v_1) + pn + s = p_i n + s_i. \tag{3}$$

If $e \in E_L(n)$, we have $v_2 - v_1 + s = s_i$ and $u_2 - u_1 + p = p_i$. If $e \in E_C(n) - E_L(n)$, we have $v_2 - v_1 + s - s_i = (p_i - u_2 + u_1 - p)n$, where $p_i - u_2 + u_1 - p = \pm 1$.

We can now prove a simple structural lemma on the relationship between circulant graphs and lattice graphs:¹

¹ For sets A, B we use the notation $A \times B$ to denote $\{\{a, b\} : a \in A, b \in B\}$.

Lemma 1.

$$E_C(n) - E_L(n) \subseteq R(n) \times L(n)$$

$$n, E_C(n) - E_L(n)$$

As examples of this lemma note that, for $C_{2n}^{1,n}$ (Figure 1)

$$E_C(n) - E_L(n) = \left\{ \{(0, 0), (1, n - 1)\}, \{(1, 0), (0, n - 1)\} \right\}$$

while for $C_{3n+1}^{1,n}$ (Figure 3)

$$E_C(n) - E_L(n) = \left\{ \{(0, 0), (2, n)\}, \{(1, 0), (0, n-1)\}, \{(2, 0), (2, n)\}, \{(2, 0), (1, n-1)\} \right\}$$

Suppose edge $e = \{(u_1, v_1), (u_2, v_2)\} \in E_C(n) - E_L(n)$. The analysis above gives us $|v_2 - v_1| \geq n - s_{max}$. But on the other hand, from Definition 4, if $e \notin L(n) \times R(n)$, $|v_2 - v_1| \leq n - s_{max} - 1$. Thus $e \in R(n) \times L(n)$.

For the constant property, we show that the condition that an edge $e = \{(u_1, v_1), (u_2, n - v_2)\} \in E_C(n) - E_L(n)$ does not depend on n , where $(u_1, v_1) \in L(n)$ and $(u_2, n - v_2) \in R(n)$. Note that v_2 can be negative and $v_1 \leq s + s_{max} - 1, |v_2| \leq s_{max}$.

There are four cases:

1. e is a jump from (u_1, v_1) by $p_i n + s_i$ and does not cross vertex $(p - 1, n + s - 1)$:
We will have $n \pm n - v_1 - v_2 = s_i$. Because $v_1 \leq s + s_{max} - 1, |v_2| \leq s_{max}$, the $\pm n$ term must be $-n$, $s_i + v_1 + v_2 = 0$ and $u_2 - u_1 = p_i - 1$. The fact that $e \in E_C(n) - E_L(n)$, is therefore independent of n .
2. e is a jump from (u_1, v_1) by $p_i n + s_i$ and crosses vertex $(p - 1, n + s - 1)$:
We will have $n \pm n + s - v_1 - v_2 = s_i$. The $\pm n$ term must be $-n$, $s - v_1 - v_2 = s_i$, and $p_i - u_2 + u_1 - p = 1$. The fact that $e \in E_C(n) - E_L(n)$, is therefore independent of n .
3. e is a jump from $(u_2, n - v_2)$ and does not cross vertex $(p - 1, n + s - 1)$:
We have $n \pm n + v_1 + v_2 = s_i$. The $\pm n$ term must be $-n$, $v_1 + v_2 = s_i$, and $u_1 - u_2 = p_i + 1$. The fact that $e \in E_C(n) - E_L(n)$, is therefore independent of n .
4. e is a jump from $(u - 2, n - v_2)$ and crosses vertex $(p - 1, n + s - 1)$:
We have $n \pm n + v_1 + v_2 + s = s_i$. The $\pm n$ term must be $-n$, $v_1 + v_2 + s = s_i$, and $u_1 - u_2 + p = p_i + 1$. The fact that $e \in E_C(n) - E_L(n)$, is therefore independent of n .

All conditions above are independent of n . So the edge set $E_C(n) - E_L(n)$ is constant (independent of n).

We have just seen that C_n can be built from L_n using a constant set of edges. We will now see that L_{n+1} can also be built from L_n using a constant set of edges.

Lemma 2.

$$E_L(n + 1) - E_L(n) \subseteq (R(n + 1) - R(n)) \times (R(n) \cup R(n + 1)).$$

..... n $E_L(n + 1) - E_L(n)$

As examples of this lemma note that, for $C_{2n}^{1,n}$ (Figure 1)

$$E_L(n + 1) - E_L(n) = \left\{ \{(0, n - 1), (0, n)\}, \{(1, n - 1), (1, n)\}, \{(0, n), (1, n)\} \right\}$$

while for $C_{3n+1}^{1,n}$ (Figure 3)

$$E_L(n + 1) - E_L(n) = \left\{ \{(0, n), (1, n)\}, \{(0, n - 1), (0, n)\}, \{(0, n), (2, n + 1)\} \right. \\ \left. , \{(1, n - 1), (1, n)\}, \{(2, n), (2, n + 1)\}, \{(1, n), (2, n)\} \right\}$$

Equations (2),(3) do not depend on n when $e = \{(u_1, v_1), (u_2, v_2)\} \in E_L(n)$. Thus if $e \in E_L(n + 1)$ and both vertices of e are in $V_L(n)$, $e \in E_L(n)$. So if $e \in E_L(n + 1) - E_L(n)$, e must contain one vertex in $V_L(n + 1) - V_L(n) = R(n + 1) - R(n)$. Furthermore we have $|v_2 - v_1| \leq s_{max}$ from the equations, which means the other node is in $R(n) \cup R(n + 1)$.

For the constant property, from Equations (2) and Equation (3), $e(n) = \{(u_1, n - v_1), (u_2, n - v_2)\} \in E_L(n)$ does not depend on n . If $e(n) \in E_L(n) - E_L(n - 1)$, $e(n)$ contains at least one vertex in $R(n) - R(n - 1)$ which directly implies that $e(n + 1)$ contains one vertex in $R(n + 1) - R(n)$. Thus $e(n) \in E_L(n) - E_L(n - 1)$ does not depend on n . So $E_L(n + 1) - E_L(n)$ is constant. \square

3 Spanning Trees

In this section \mathcal{A} objects are spanning trees and our problem is to count the number of spanning trees in C_n . Recall that, given a graph $G = (V, E)$, a spanning tree $T \subseteq E$ is a subset of the edges that forms a connected acyclic graph.

Let $p, s, p_1, p_2, \dots, p_k$ and s_1, s_2, \dots, s_k be given nonnegative integral constants with $\forall i, p_i \leq p$. Set $C_n = C_{pn+s}^{p_1n+s_1, p_2n+s_2, \dots, p_kn+s_k}$ to be the circular graph, $L_n = L_{pn+s}^{p_1n+s_1, p_2n+s_2, \dots, p_kn+s_k}$ to be the lattice graph, and $T(n) = T(C_n)$ to be the number of spanning trees in C_n .

In [11] tools were developed for constructing recurrence relations on structures of circulant graphs. The difficulty in extending that result to circulants was the lack of some type of recursive decomposition of non-constant-jump circulants. Given the Lattice graph representation of circulant graphs of Lemma 1 and the recursive construction of Lattice graphs implied by Lemma 2 we can now plug these facts into the tools developed in [11] and develop recurrence formulas for $T(n)$. Since the proofs are rather straightforward and follow those of [11] we do not give them here. In the appendix we will show how to use these techniques to rederive the exact solution for $T\left(C_{2n}^{1,n}\right)$.

Let T be a spanning tree of circulant graph C_n . Removing all edges of $E_C(n) - E_L(n)$ from T leaves a forest $T \cap E_L$ in Lattice graph L_n . Lemma 1 tells us that all endpoints of edges in $E_C(n) - E_L(n)$ are in $L(n) \cup R(n)$, so $T \cap E_L$ is a legal forest of L_n . This motivates the following definition of $Par(W)$:

Definition 5. For $n > s + 2s_{max} - 1$, let $W(n) = L(n) \cup R(n)$. Let $Par(W)$ be the set of all partitions of W ,
 $Par(\{1, 2, 3\}) = \{ \{\{1, 2\}\{3\}\}, \{\{1, 3\}\{2\}\}, \{\{2, 3\}\{1\}\}, \{\{1\}\{2\}\{3\}\}, \{\{1, 2, 3\}\} \}$.
 A legal forest F of L_n is called a *legal forest* of $W(n)$ if F is a forest of $W(n)$.
 $\mathcal{P} = Par(W(n))$ is the set of all partitions of $W(n)$.
 $C(F)$ is the classification of F , where $X \in \mathcal{P}$ if and only if $\forall u, v \in W(n), u, v$ are connected in F if and only if $u, v \in X$.
 $T_X(n) = |\{F : F \text{ is a legal forest of } L_n \text{ and } C(F) = X\}|$

Note: The reason for requiring $n > s + 2s_{max} - 1$ is to guarantee that $L(n) \cap R(n) = \emptyset$.
 We are now interested in how to reconstruct spanning trees from legal forests. Define

Definition 6. $\mathcal{S} = \{S : S \subseteq (E_C - E_L)\}$.

Note that, given a legal forest F , it may not always be possible to find $S \in \mathcal{S}$ such that $F \cup S$ is a spanning tree of C_n . We make the following straightforward observation

Lemma 3. For any legal forest F, F' of L_n , $C(F) = C(F')$ if and only if $S \in \mathcal{S}$ such that $F \cup S$ and $F' \cup S$ are spanning trees of C_n .

This permits the following definition

Definition 7. For $X \in \mathcal{P}$ and $S \in \mathcal{S}$, let

$$\alpha_{S,X} = \begin{cases} 1: & \text{if } S \cup F \text{ is a spanning tree of } C_n \text{ and } C(F) = X \\ 0: & \text{otherwise} \end{cases}$$

$$\beta_X = \sum_{S \in \mathcal{S}} \alpha_{S,X} \quad \beta = (\beta_X)_{X \in \mathcal{P}}$$

Let $\beta = (\beta_X)_{X \in \mathcal{P}}$.

The crucial observation in the above definitions is that Lemma 3 implies that $\alpha_{S,X}$ is a function of n and can be easily evaluated just by looking at S and X .

As an example, suppose we are given $C_{2n}^{1,n}$ and its $L_{2n}^{1,n}$ and, for some n , F is a legal forest of $L_{2n}^{1,n}$ with $C(F) = X = \{(0, 0), (0, n - 1), (1, 0)\}, \{(1, n - 1)\}$. That is, F has exactly two connected components partitioning the nodes in

$W(n) = L(n) \cup R(n)$; one of the components contains $(0, 0), (0, n - 1), (1, 0)$ and the other contains $(1, n - 1)$. Now, if $S = \{(0, 0), (1, n - 1)\}$ then $\alpha_{S,X} = 1$ since the single edge in S connects the two components to form a spanning tree while if $S = \{(0, n - 1), (1, 0)\}$ then $\alpha_{S,X} = 0$ since the single edge in S creates a cycle in the component containing $(0, 0), (0, n - 1), (1, 0)$.

Since, by definition, every spanning tree of C_n is uniquely decomposable into a legal forest F of L_n plus some $S \in \mathcal{S}$ we immediately find

Lemma 4.

$$T(C_n) = \sum_{X \in \mathcal{P}} \left(\sum_{S \in \mathcal{S}} \alpha_{S,X} \right) T_X(n) = \sum_{X \in \mathcal{P}} \beta_X T_X(n). \tag{4}$$

Letting $\mathbf{T}(L_n)$ be the column vector $(T_X(n))_{X \in \mathcal{P}}$, this can also be written as $T(C_n) = \beta \cdot \mathbf{T}(L_n)$.

So far we have only shown that the number of spanning trees of a circulant graph is a linear combination of the number of different legal forests of the associated lattice graph. We will now show the the number of different legal forests can be written as a system of linear recurrences in n . The main observation is the following lemma:

Lemma 5. $F \dots L_{n+1} \dots U = F \cap (E_L(n + 1) - E_L(n)).$
 $F - U \dots L_n.$

Note that this lemma implies that every legal forest of L_{n+1} can be built from a legal forest of L_n . To continue we will need the following observation:

Lemma 6. $F, F' \dots L_n \dots C(F) = C(F').$
 $U \subseteq E_L(n + 1) - E_L(n).$
 $- F \cup U \dots L_{n+1} \dots F' \cup U \dots L_n$
 $- \dots F \cup U \dots F' \cup U \dots L_{n+1} \dots C(F \cup U) = C(F' \cup U).$

Before continuing we should emphasize a subtle point concerning the classification of a \dots in L_n , which is that $\dots n$. For example, in lattice graph $L_{2n}^{1,n}$, when $n = 3$, a $\dots D$ with classification $C(D) = \{(0, 0), (0, n - 1)\}, \{(1, 0), (1, n - 1)\}$ implies that D has two components with one containing nodes $(0, 0)$ and $(0, 2)$ and the other containing nodes $(1, 0)$ and $(1, 2)$. Now suppose $n = 4$ with no edges added to D , in the new lattice graph, the new forest D' contains four components, which means $C(D') = \{(0, 0)\}, \{(0, n - 1)\}, \{(1, 0)\}, \{(1, n - 1)\}$. When calculating how adding vertices and edges to legal forests in L_n changes them into different legal forests in L_{n+1} we must take account of this fact.

Lemma 6 permits the next definition

Definition 8. $X, X' \in \mathcal{P} \dots U \subseteq E_L(n + 1) - E_L(n) \dots$
 $\gamma_{X',X,U} = \begin{cases} 1 : \dots U \dots F \dots C(F) = X' \dots F' \\ \dots C(F') = X' \\ 0 : \dots \dots \end{cases}$

$$\alpha_{X',X} = \sum_{U \subseteq E_L(n+1) - E_L(n)} \gamma_{X',X,U} \quad A = (a_{X',X})_{X',X \in \mathcal{P}}$$

$$\beta = \dots$$

Note: As in the observation following Lemma 7 we point out that the values of $\gamma_{X',X,U}$ and thus $\alpha_{X',X}$ and A are *independent* of n . It is therefore possible to mechanically calculate all of the β_X and $\alpha_{X',X}$.

Combining Lemmas 5 and 6 then yield

Lemma 7. $\forall X' \in \mathcal{P}$,

$$T_{X'}(n+1) = \sum_{X \in \mathcal{P}} a_{X',X} T_X(n) \quad \mathbf{T}(L_{n+1}) = \mathbf{A}\mathbf{T}(L_n)$$

Combining everything in this section gives our main theorem on spanning trees of circulant graphs, proving Theorem 1 for the case $\mathcal{A} = \dots$

Theorem 2. $T(n) = \dots C_n \dots \mathcal{P} = \dots Par(L(n) \cup R(n)), T_X(n) \dots$ legal forests $\dots X \dots \mathbf{T}(L(n)) \dots (T_X(n))_{X \in \mathcal{P}} \dots n \geq s + 2s_{max} - 1,$

$$T(C(n)) = \beta \cdot \mathbf{T}(L(n))$$

$$\mathbf{T}(L(n+1)) = \mathbf{A}\mathbf{T}(L(n))$$

$$\beta = \dots A = \dots$$

This theorem implies that $T(C(n))$ satisfies a linear recurrence relation with constant coefficients of order rank of the matrix A .

Since the \dots of matrix A is $|\mathcal{P}| = B(p(s+2s_{max})+s)$ where $B(m)$ is the Bell number² of order m the order of the recurrence is at most $B(p(s+2s_{max})+s)$.

References

1. G. Baron, H. Prodinger, R. F. Tichy, F. T. Boesch and J. F. Wang. "The Number of Spanning Trees in the Square of a Cycle," *Fibonacci Quarterly*, **23.3** (1985), 258-264.
2. S. Bedrosian. "The Fibonacci Numbers via Trigonometric Expressions," *J. Franklin Inst.* **295** (1973), 175-177.
3. J.-C. Bermond, F. Comellas, D.F. Hsu. "Distributed Loop Computer Networks: A Survey", *Journal of Parallel and Distributed Computing*, **24**, (1995) 2-10.
4. N. Biggs, *Algebraic Graph Theory*, London: Cambridge University Press, Second Edition, 1993.

² $B(m)$ counts the number of set partitions of m items.

5. F. T. Boesch, J. F. Wang. "A Conjecture on the Number of Spanning Trees in the Square of a Cycle," In: *Notes from New York Graph Theory Day V*, New York: New York Academy Sciences, 1982. p. 16.
6. F. T. Boesch, H. Prodinger. "Spanning Tree Formulas and Chebyshev Polynomials," *Graphs and Combinatorics*, **2**, (1986), 191-200.
7. Graham R. Brightwell and Peter Winkler. "Note on Counting Eulerian Circuits," *lanl.arXiv.org*, **cs.CC/0405067** (May 19 2004).
8. C. J. Colbourn. *The combinatorics of network reliability*, Oxford University Press, New York, (1987).
9. D. Cvetkovič, M. Doob, H. Sachs. *Spectra of Graphs: Theory and Applications, Third Edition*, Johann Ambrosius Barth, Heidelberg, (1995).
10. M. J. Golin, Y.P. Zhang. "Further applications of Chebyshev polynomials in the derivation of spanning tree formulas for circulant graphs," in *Mathematics and Computer Science II: Algorithms, Trees, Combinatorics and Probabilities*, 541-552. Birkhauser-Verlag. Basel. (2002)
11. M. J. Golin and Yiu Cho Leung. "Unhooking Circulant Graphs: A Combinatorial Method for Counting Spanning Trees and Other Parameters," To appear in the Proceedings of the 30'th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'2004) (2004).
12. F.K. Hwang. "A survey on multi-loop networks," *Theoretical Computer Science* **299** (2003) 107-121.
13. G. Kirchhoff. "Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird," *Ann. Phys. Chem.* **72** (1847) 497-508.
14. D. J. Kleitman, B. Golden. "Counting Trees in a Certain Class of Graphs," *Amer. Math. Monthly*, **82** (1975), 40-44.
15. John P. McSorley. "Counting structures in the Möbius ladder," *Discrete Mathematics* **184** (1998), 137-164 .
16. M. Mihail and P. Winkler. "On the number of Eulerian orientations of a graph," *Algorithmica*, **16** (1996) 402-414.
17. Martin Rubey. *Counting Spanning Trees*, Diplomarbeit, Universität Wein, Mai 2000.
18. J. Sedlacek. "On the skeletons of a Graph or Digraph," in *R.K. Guy et al., (Eds.), Combinatorial Structures and their Applications*, (387-391.) 1970.
19. J. A. Sjogren. "Note on a formula of Kleitman and Golden on spanning trees in circulant graphs," *Proceedings of the Twenty-second Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Congr. Numer. **83** (1991), 65-73.
20. L.G. Valiant. "The complexity of enumeration and reliability problems," *SIAM J. Comput.*, **8** (1979) 410-421.
21. R. Vohra and L. Washington. "Counting spanning trees in the graphs of Kleitman and Golden and a generalization," *J. Franklin Inst.*, **318** (1984), no. 5, 349-355
22. Q.F. Yang, R.E. Burkard, E. Cela and G. Woeginger. "Hamiltonian cycles in circulant digraphs with two stripes," *Discrete Math.*, **176** (1997) 233-254.
23. X. Yong, Talip, Acenjian. "The Numbers of Spanning Trees of the Cubic Cycle C_N^3 and the Quadruple Cycle C_N^4 ," *Discrete Math.*, **169** (1997), 293-298.
24. X. Yong, F. J. Zhang. "A simple proof for the complexity of square cycle C_p^2 ," *J. Xinjiang Univ.*, **11** (1994), 12-16.
25. Y. P. Zhang, X. Yong, M. J. Golin. "The number of spanning trees in circulant graphs," *Discrete Math.*, **223** (2000) 337-350.

Adaptive Spatial Partitioning for Multidimensional Data Streams^{*}

John Hershberger¹, Nisheeth Shrivastava², Subhash Suri², and Csaba D. Tóth²

¹ Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070, USA,
and (by courtesy) Computer Science Dept., University of California, Santa Barbara
`john_hershberger@mentor.com`

² Computer Science Dept., University of California, Santa Barbara, CA 93106, USA
{`nisheeth, suri, toth`}@cs.ucsb.edu

Abstract. We propose a space-efficient scheme for summarizing multi-dimensional data streams. Our scheme can be used for several geometric queries, including natural spatial generalizations of well-studied single-dimensional queries such as icebergs and quantiles.

1 Introduction

Many current and emerging technologies generate *continuous*, data feeds at very high rates that require continuous processing. There are many sources of such data: sensor networks monitoring environments such as industrial complexes, hazardous sites, or natural habitats; scientific instruments collecting astronomical, geological, ecological, or weather-related observation data; as well as measurement data for monitoring complex distributed systems such as the Internet. Despite their obvious differences, there are many similarities and commonalities among these data: they are all multidimensional; they have large volume; much of the data is routine and uninteresting to the user; and the user typically is interested in detecting unusual patterns or events.

As a result, there has been enormous interest in designing data-processing algorithms that work over continuous data streams; these algorithms look at the data only once and in a fixed order (determined by the stream-arrival pattern). We assume that the algorithm has access to only a small amount of memory, which is significantly smaller than the size of the full stream, n . The algorithm must therefore represent the data in a concise “summary” that can fit in the available memory and can be used to provide guaranteed-quality approximate answers to user queries.

We propose a novel, versatile, and space-efficient scheme for summarizing the spatial distribution of a d -dimensional point stream. Our *adaptive spatial partitioning* (ASP) scheme is fully deterministic and combines three basic data

^{*} Research by the last three authors was partially supported by National Science Foundation grants CCR-0049093 and IIS-0121562.

structures: a quadtree-like structure and two search trees. Specifically, given a stream of d -dimensional points $p_1, p_2, \dots, p_n, \dots$, drawn from the domain $[0, R]^d$, the ASP is an $O(\frac{1}{\epsilon} \log R)$ size data structure, where ϵ is a user-specified parameter. It can answer a variety of geometric queries with a guaranteed error bound (and no low-frequency false positives). Some examples of such queries are \dots and \dots and rank and range queries (please see Section 3 for details).

Our ASP scheme also works for weighted point streams, where each point has a (positive) weight, and the approximation error is measured as a fraction of the total weight. It extends easily to the \dots model of a data stream, with an $O(\log \epsilon n)$ factor blowup in space, where n is the window size. All space bounds in this paper (including references to previous work) assume the \dots model of computation, where each counter value can be stored in a single word. If the space is measured in bits, then the size of our data structure grows by a factor of $\log \epsilon n$.

1.1 Related Previous Work

One of the best-studied problems in data streams is finding \dots ; variants of this problem are also called \dots [13] in databases and \dots [12] in networking. The best deterministic stream algorithm for this problem is the \dots scheme of Manku and Motwani [18], which uses $O(\frac{1}{\epsilon} \log \epsilon n)$ space. (There are simpler schemes, using $O(1/\epsilon)$ space, by Misra and Gries [21], Karp, Shenker, and Papadimitriou [17] and Demaine, López-Ortiz, and Munro [11], but they suffer from the false positive problem—they cannot guarantee any lower bound on the frequency of the items found.) Cormode et al. [7] consider a hierarchical version of the one-dimensional heavy hitter problem using $O(\frac{h}{\epsilon} \log \epsilon n)$ space, where h is the depth of the hierarchy.

Approximate quantile computation is another well-studied problem in data streams. For this problem, a classical result of Munro and Paterson [22] shows that any algorithm for computing \dots quantiles in p passes requires $\Omega(n/p)$ space; thus, any single-pass, sublinear-space algorithm must resort to approximation. The best deterministic stream algorithm for ϵ -approximate quantiles is due to Greenwald and Khanna [15], and uses $O(\frac{1}{\epsilon} \log \epsilon n)$ space. Just recently, Arasu and Manku [3] have extended the Greenwald-Khanna algorithm to the sliding window model; their algorithm uses $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log n)$ space, where n is the window size.

Multidimensional stream processing is less well-developed, but there is growing literature on several interesting problems. For instance, Cormode and Muthukrishnan [8], Agarwal, Har-Peled, and Varadarajan [1], and Hershberger and Suri [16] have proposed stream algorithms for extremal geometric structures, such as convex hull, diameter, width, etc. The challenge in maintaining such objects is that any data point can change the output dramatically and has to be accounted for; none of these schemes summarizes the \dots of the stream. Charikar, O’Callaghan, and Panigrahy [6] have considered the k -median problem; Thaper et al. [25] have considered multidimensional histograms; both use randomized techniques. In [12], Estan, Savage, and Varghese propose the no-

tion of hierarchically defined multidimensional for network monitoring. However, their solutions are offline, and so do not work in the streaming model. is an intuitively appealing idea for summarizing multidimensional point streams. Indeed, the scheme of Manku and Motwani [18] uses a random sample of size $O(\frac{1}{\varepsilon} \log \varepsilon n)$ to maintain single-dimensional heavy-hitters with high probability. If one wants high probability frequency estimates of any interval, then we need to use the concept of ε A $\Theta(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})$ size random sample is an ε -approximation with high probability [26]. A randomized algorithm of Manku, Rajagopalan, and Lindsay [19, 20] can maintain an ε -approximate ϕ -quantile for any fixed $\phi \in (0, 1)$ in $O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})$ space. A very recent randomized algorithm of Suri, Tóth, and Zhou [24] can maintain an ε -approximation of size $O(\frac{1}{\varepsilon} \log^{2d+1} \frac{1}{\varepsilon})$ for axis-aligned boxes in \mathbb{R}^d with constant probability. This algorithm is, however, only of theoretical interest because it requires substantial working space (i.e., solving $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \log(\varepsilon n))$ linear programs in $O(\frac{1}{\varepsilon} \log^{2d+1} \frac{1}{\varepsilon})$ variables).

2 Adaptive Spatial Partitioning

In this section, we describe our summary scheme and establish its space and update complexity. We describe the scheme for 2-dimensional points for ease of exposition, but the ideas generalize readily to d dimensions, for any fixed $d \in \mathbb{N}$, with an $O(2^d)$ factor increase in space.

We assume that the input stream consists of points from the universe $[0, R]^2 \subset \mathbb{R}^2$. The point coordinates can have arbitrary precision, but our resolution is limited to 1; that is, only the most significant $\log R$ bits matter. (In some applications, such as network monitoring, the space is naturally discrete: $R = 2^{32}$ is the size of the IP address space. In other, non-discrete domains, such as astronomy or geological data, the resolution may be user-defined.) We make no assumption about the order in which points arrive. Without loss of generality, we assume that the i^{th} point arrives at time i . We denote the number of elements seen so far by n .

2.1 Data Structure

Our summary is based on a hierarchical decomposition of the space, which is represented in a data structure called the Adaptive Spatial Partitioning tree. It is complemented by two auxiliary structures, a min-heap and a point location tree, that are used to perform merge and point location operations efficiently. We now describe the details of these data structures.

The tree (.) T is a 4-ary (in d dimensions, a 2^d -ary) tree. Each node $v \in T$ is associated with a grid-aligned region of the plane. We denote this region by B_v , and call it the v . We also maintain a counter for v , denoted (v). The box associated with the root r of T is the entire universe $B_r = [0, R]^2$. The boxes follow the tree hierarchy: $u \in (v)$ implies $B_u \subset B_v$ and the side length of B_u is at most

half the side length of B_v . Because the maximum box size shrinks by half at every level, and the minimum resolution is one, the depth of the ASP tree is at most $\lceil \log R \rceil$.

The ASP tree adapts to changes in the distribution of data over time. High density regions require a fine box subdivision, while for low density regions, a rougher subdivision is sufficient. For every point insertion, we increment n , the total stream size, and the counter of the smallest box B containing the new element. Thus every point is counted at exactly one node. If the counter of a node is above a threshold αn , for a parameter α to be specified later, then we refine B by subdividing it into smaller boxes. Since n keeps increasing, the sum of the counts of a box B and its children may later be below this threshold, and so the refinement is no longer necessary. We can merge B , by merging it with its children, to save memory.

The ASP tree is a compressed version of the standard quad-tree; in a quad-tree, the four children of a node have boxes with half the side length of the parent box, but in the ASP tree, though every node has four children, the boxes associated with the children may be much smaller. We maintain the invariant that the four children’s boxes partition the box of v or one of its quad-tree descendants. This implies that we may only unrefine nodes with at most one non-leaf child. We call such nodes *mergeable*.

While refinement is directly linked with the point being inserted, unrefinement is not. Naïvely, at every point arrival, we should check all the mergeable nodes for possible unrefinement. Instead, we use a standard **min-heap** to maintain a priority queue of all the mergeable nodes; the top of the heap is the node that will become unrefinable earliest. The nodes in the merge heap M are exactly the internal nodes with a non-leaf child in T . The key of each node v is

$$key(v) = n \cdot \text{depth}(v) + \sum_{x \in \text{children}(v)} key(x).$$

The merge heap can be updated after an insert or delete in $O(\log |M|) = O(\log |T|)$ time.

The insertion of every point p requires searching in the ASP tree for the smallest box containing p . Using the ASP tree, the point location might take $O(\log R)$ time, the depth of the ASP tree. We can improve the time spent on point location among the $|T|$ nodes of the ASP tree to the optimal $O(\log |T|)$ per insertion with an auxiliary structure C , a standard binary search tree defined on top of T . Every node $u \in C$ corresponds to a connected subtree T_u of T . If T_u has more than one node, then u stores an approximate centroid edge e_u of T_u that divides T_u into two roughly balanced parts; that is, each subtree must have size at least $c_d \cdot |T_u|$, for some constant c_d . The two subtrees, in turn, correspond to the two children of u in C .

The query time to find the minimal box of T containing a point q is the depth of $C(T)$, which is $O(\log |T|)$ because the size of the subtrees decreases by a constant factor with the depth in C . Every internal node $u \in C$ corresponds to an edge vw of T which, in turn, corresponds to the pair of boxes $B_w \subset B_v$

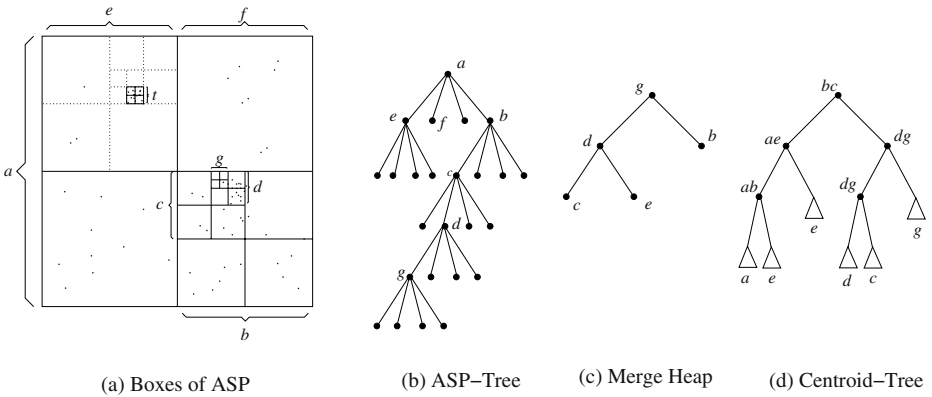


Fig. 1. The boxes of the ASP for a distribution of input data points, the ASP tree, the corresponding merge heap, and centroid tree. The boxes in the ASP are drawn with solid lines. Dotted lines indicate boxes that were in the ASP tree at some time but have been removed by unrefinement operations. Box B_t was once the great grandchild of B_e , but in our current ASP tree, the children of t are stored as direct children of e . The nodes b, c, d, e, g are in the merge heap, since each has at least three leaf children

in \mathbb{R}^d . At node u , we test $q \in B_w$ and continue searching q in the left or right subtree of u accordingly. The search returns a leaf of C , which corresponds to a node $v(q) \in T$ and a box $B_{v(q)}$. The tests performed along the search prove that $q \in B_{v(q)}$, but $q \notin B_w$ for any $B_w \subset B_{v(q)}$, $w \in T$. Figure 1 shows an example of our data structure.

2.2 Maintaining the ASP

We initialize the ASP with a one-node ASP tree, with box B_r corresponding to the root, $\text{count}(r)$ set to zero, an empty merge heap, and a one-node approximate centroid tree C .

When a new point p arrives, we find the smallest box in the ASP tree containing p and increment its count by one. This changes at most two keys in the merge heap (of the box and its parent), and we update the heap accordingly. For the majority of point insertions, this is all the work needed to update the ASP. Occasionally, however, we need to adjust the tree structures as well. We use two basic operations to maintain our summary, refine and unrefine , described below:

The refine operation occurs when the counter of a node v exceeds a threshold value αn . That is, if $\text{count}(v) > \alpha n$ and the box B_v is not a unit (minimum resolution) box, then we split B_v into four congruent boxes, and add edges from node v to the new children corresponding to the four half-size sub-boxes. (If v already had four children that did not cover B_v , then those children are detached from v and re-attached as children of one of v 's four newly-created children.) As long as v 's children cover B_v , the new point only increments the counter of one of the descendants of v , and $\text{count}(v)$ remains fixed.

The *unrefinement* operation is performed when the key of a node in the merge heap is exceeded by the *split threshold*. We set the merge threshold to be half of the split threshold, though this is tunable. Specifically, if v is a node in M and

$$key(v) < \frac{\alpha n}{2}, \tag{1}$$

then we merge the children of v into v , and set $key(v) := key(v)$. (Recall that $key(v)$ is defined as $key(v) + \sum_{x \in children(v)} key(x)$.) That is, we simply redirect the parent pointers of v 's grandchildren to point to v , thereby deleting v 's children, and fold the counters of v 's children into v . It is worth noting that the updated v has at most four children, because no node in M can have more than four grandchildren.

Due to lack of space, we omit from this extended abstract the details of how these operations affect the merge heap and centroid tree.

2.3 Complexity Analysis

We now prove that the space complexity of our data structures is $O(\frac{1}{\alpha})$, where α is the threshold parameter defined above. We also show that the amortized update time for point insertion is $O(\log \frac{1}{\alpha})$.

Theorem 1. *Let α be the threshold parameter. The space complexity of the data structures is $O(\frac{1}{\alpha})$, and the amortized update time for point insertion is $O(\log \frac{1}{\alpha})$.*

We argue that the number of internal nodes in T is $O(\frac{1}{\alpha})$. Since every node has at most four (2^d , in \mathbb{R}^d) children, this implies that the space complexity is $O(\frac{1}{\alpha})$. We first bound the number of nodes in the merge heap. No node in M satisfies the Merge Inequality (1)—otherwise, the unrefinement operation would have deleted that node from M . Thus, for each $v \in M$, we must have $key(v) \geq \frac{\alpha n}{2}$. The value $key(w)$ of a node $w \in T$ contributes to at most two keys, and so we have the following inequality:

$$\begin{aligned} |M| \cdot \frac{\alpha n}{2} &\leq \sum_{v \in M} key(v) < \sum_{v \in T} key(v) = \sum_{v \in T} \left(key(v) + \sum_{x \in children(v)} key(x) \right) \\ &\leq 2 \sum_{w \in T} key(w) = 2n. \end{aligned}$$

Thus the merge heap has at most $2n / (\frac{\alpha n}{2}) = 4/\alpha$ nodes.

Any internal node that is not in M has at least two children that are also internal nodes of T . These are branching nodes of the tree on paths to nodes in the merge heap. This implies that the total number of internal nodes in T is at most $8/\alpha$. This completes the proof.

We show next that the amortized updates require $O(\log \frac{1}{\alpha})$ time.

Theorem 2. *Let α be the threshold parameter. The amortized update time for point insertion is $O(\log \frac{1}{\alpha})$, and the amortized update time for point deletion is $O(\log \frac{1}{\alpha})$. The space complexity of the data structures is $\Omega(\frac{1}{\alpha})$.*

When a new point p arrives, we find the smallest box in the ASP tree containing p in $O(\log |T|) = O(\log \frac{1}{\alpha})$ time using the point location tree. After the insertion, we perform at most one refinement, which requires $O(1)$ time in the ASP tree, and update M , which requires $O(\log |M|) = O(\log \frac{1}{\alpha})$ time.

We can also update the approximate centroid tree in amortized $O(\log \frac{1}{\alpha})$ time. Suppose e is the exact centroid edge of a tree T_u . Then e remains an approximate centroid as long as fewer than $\kappa|T_u|$ refinements and unrefinements modify T_u , for some constant κ . After $\Omega(|T_u|)$ events, it may be necessary to recompute the exact centroid tree of T_u , which takes $O(|T_u|)$ time. Thus, rebalancing the approximate centroid tree requires $O(k)$ time after k refinements or unrefinements at each of the $O(\log |T|) = O(\log \frac{1}{\alpha})$ levels of C . Hence the amortized cost of the rebalancing is $O(\log \frac{1}{\alpha})$.

To bound the total number of unrefinements, we note that each refinement increases the number of nodes in T by 4, and each unrefinement decreases the number of nodes by the same amount. The total number of nodes is positive, and so there can be no more unrefinements than refinements. Like a refinement, an unrefinement takes $O(\log \frac{1}{\alpha})$ amortized time, and so the proof is complete.

3 Applications and Extensions

Our approximations will have an absolute εn error. Just as in iceberg queries and quantile summaries, this error guarantee is the best possible for schemes using roughly $O(1/\varepsilon)$ space. Specifically, Alon, Matias, and Szegedy [2] prove that, given a data stream in a domain of size R , any (randomized or deterministic) algorithm that approximates the frequency of the most frequent item to within a constant error must use $\Omega(R)$ space. Because hot spots and range counting problems are strict generalizations of iceberg and quantile summaries, these approximation quality lower bounds apply to ASP as well.

3.1 Applications

Hot Spots. We define a ε -hot spot in \mathbb{R}^d as an integer unit cube containing at least εn points of the stream. Hot spots are a natural spatial analogue of iceberg queries: in multidimensional data streams, none of the individual points may occur with large frequency, yet a closely clustered set of points may signal unusual or suspicious activity. Formally, an ε -hot spot is defined as $\prod_{i=1}^d [x_i, x_i + 1)$ where each x_i is an integer in the range $\{0, 1, \dots, R - 1\}$. Hypercubes of larger granularity or more general rectangular templates are easy to handle as well.

To compute ε -hot spots, we maintain a d -dimensional ASP with parameter $\alpha = \frac{\varepsilon}{2 \log R}$, and report every leaf node that corresponds to a unit cube and whose counter exceeds $\frac{\varepsilon}{2}n$. By choosing $\alpha = \frac{\varepsilon \delta}{\log R}$, for any $\delta \in (0, 1)$, we can track hot spots where each cell is guaranteed to contain at least $(1 - \delta)\varepsilon n$ points. One can also extract hypercube regions of larger granularity from our data structure. Due to lack of space, we omit the detailed analysis.

Cold Spots. A natural definition of the cold spot is the box with at most ϵn points. Since the leaf cells of our ASP are precisely the boxes with fewer than ϵn points, we can track the cold spot by, say, keeping these boxes sorted in descending order of size to population ratio.

Hierarchical Hot Spots. Hierarchical hot spots are a spatial generalization of the heavy hitters, introduced by Cormode et al. [7] for 1-dimensional data. In a rooted tree representation of the data, the hierarchical heavy hitters are defined recursively in a bottom-up order: each node whose subtree has frequency more than ϵn , not counting the frequencies of its heavy hitter descendants, is a heavy hitter.

In order to maintain hierarchical hot spots using ASP, we add a counter $c(v)$ at each node v to track the total population of all the descendants of v . That is, $c(v) = \sum_{w \in \text{descendants}(v)} c(w)$. This increases the worst-case per-point update time of the ASP from $O(\log \frac{1}{\alpha})$ to $O(\log R)$, which is the depth of T . To determine the heavy hitters at each level of T , we set $\alpha = \frac{\epsilon}{2 \log R}$ and report every node of T such that the value $c(v) - \sum \{ c(w) : w \text{ is a descendant of } v \text{ and a heavy hitter} \}$ is above $\frac{\epsilon}{2}n$.

Rank and Range Queries. In a set of d -dimensional points, the rank of a point p is the number of points dominated by p on all coordinates. This is a natural generalization of the rank in one dimension, where the rank of an element x is the number of elements less than x . The rank and quantiles are intimately related: quantiles are elements of specified ranks; on the other hand, given the quantiles, the rank of an element can be located between two consecutive quantiles. Thus, the 1-dimensional ASP with $\alpha = \frac{\epsilon}{\log R}$ can also be interpreted as an ϵ -rank quantile query, and thus offers an alternative deterministic scheme for this well-studied problem.

Determining the rank in more than one dimension is equivalent to answering range counting queries for axis-parallel query rectangles, which are defined as $\prod_{i=1}^d [0, b_i]$. This is so because every (axis-parallel) rectangle can be expressed with 2^d grounded rectangles using inclusion-exclusion. We can use the ASP directly for multidimensional range counting with the following simple algorithm: Given a query box Q , we report the sum of all counters $c(v)$ weighted with the fraction of the volume of each box B_v lying in the query box, that is, $\sum_v c(v) \cdot \text{vol}(B_v \cap Q) / \text{vol}(B_v)$. Although the theoretical approximation quality of this simple scheme can be arbitrarily bad in the worst case, our experiments show that it performs well in practice.

With a recursive construction, we can use ASP to provide good theoretical guarantees for multidimensional range queries; however, the space bound for the construction grows as $O(\frac{1}{\epsilon} \log^{2d-1} R)$. Due to lack of space, we omit further details.

3.2 Extensions of the Streaming Model

In some applications, the data stream is infinite, but only the “recent” portion of the stream is relevant. The sliding window model of data streams is motivated

by these applications [4, 10, 23]. Let n denote the size of the window; that is, we want to maintain a summary over the last n points. A point p_i that is inserted at time i is deleted at time $i + n$, which we refer to as its *sliding window lifetime*.

In the full version of the paper, we show that the ASP extends to the sliding window model, with an $O(\log \varepsilon n)$ factor increase in space, where n is the window size. The (amortized) per-point processing cost is unaffected. We also discuss extensions to weighted point streams, and a restricted class of the turnstile model.

4 Experimental Results

We implemented the adaptive spatial partitioning both for single- and multi-dimensional streams. In this section, we briefly report on some of our experimental results.

For two-dimensional adaptive spatial partitioning, we used three different data sets: *gazetteer*, which consisted of n points generated uniformly at random inside the box $[0, R]^2$; *k-peaks*, which included k Gaussian distributions concentrated around k randomly chosen centers, against a background of low density uniformly scattered points (noise); and the *gazetteer* data from UCSB’s Alexandria Digital Library (see Figure 2), which consists of the locations of 10^7 geographical features. In each case we used $R = 2^{20}$.

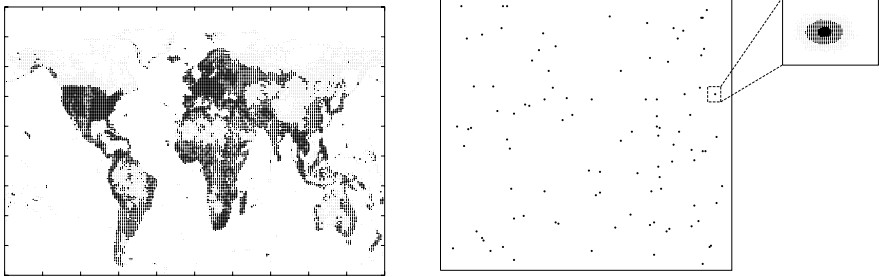


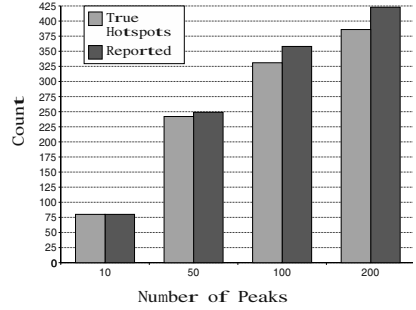
Fig. 2. Input datasets: *gazetteer* (left) and *k-peaks* with $k = 100$ (right). The gray scale reflects the density of data points in each region. The inset picture on the right figure shows a Gaussian distribution at a single peak

4.1 Memory Usage of ASP

Table 1 shows the space needed for ASP as a function of n and ε . The main point to note is that, for a fixed value of ε , the size of the ASP is essentially independent of n . It increases linearly with $\frac{1}{\varepsilon}$, as predicted by the theoretical analysis. For the *k-peaks* data the size depends more on k than ε , because ASP quickly “zooms” into the k -centers and maintains partitions only for the regions around them.

Table 1. ASP tree size for different experimental data

# points	ε	ASP tree size		
		0.01	0.001	0.0001
100-peaks	10^6	2269	3433	9457
	10^7	2237	3265	4229
	10^8	2237	3261	4061
random	10^6	5461	83093	492269
	10^7	5461	87381	387749
	10^8	5461	87381	349525
gazetteer	10^6	5965	56169	532933
	10^7	5853	55989	530585

**Fig. 3.** True versus reported hot spots

4.2 Detecting Hot Spots with ASP

Figure 3 shows how well ASP detects hot spots in the k -peaks data. In our experiment, a total of $n = 10^6$ points were streamed, and we fixed $\varepsilon = 10^{-3}$. Given a specific value of k , $10^6/(k+1)$ points were generated in each of the k Gaussian clusters, and the final $10^6/(k+1)$ points were distributed uniformly inside the square (as background noise).

Our algorithm detects all the true hot spots, and a small number of false positives as well. The false positives are within a factor of two of being hot. We also note (data not shown) that the size of the ASP tree grows essentially linearly with k , assuming that $\varepsilon < 1/k$, as is the case in these experiments.

4.3 Computing Quantiles

Although ASP is a general-purpose representation of multidimensional data, it is interesting to compare its performance on 1-dimensional data to that of problem-specific 1-dimensional algorithms. We used ASP to compute 1-dimensional quantiles, using a stream of integer values randomly chosen from the range $[1, 2^{20}]$. We compared the results with two specialized schemes for quantile computation: Greenwald-Khanna's algorithm [15] (GK) and Manku, Rajagopalan, and Lindsay's algorithm [19] (MRL).

We fix $\varepsilon = 10^{-3}$ and maintain the ASP summary for streams of size $n=10^5$, 10^6 and 10^7 . We compute the quantiles of the input at rank $\frac{i}{16}n$ for $i = [1..15]$.

Table 2. Quantile approximation for streams

n	MRL		ASP		GK	
	Size Error ($\times 10^{-4}$)	Size Error ($\times 10^{-4}$)	Size Error ($\times 10^{-4}$)	Size Error ($\times 10^{-4}$)	Size Error ($\times 10^{-4}$)	Size Error ($\times 10^{-4}$)
10^5	8334	4.69	2027	5.59	919	8.48
10^6	15155	3.27	2047	4.01	919	8.00
10^7	27475	2.35	2047	6.60	919	7.82

The error in computed rank is ($\text{trueRank} - \text{computedRank}$). We take the average of these errors for each scheme. Table 2 shows the variation of the size and the average error with n . Each error is shown as a fraction of stream size n . (The value 4.01 in the second row of ASP means the rank error was $4.01 \times 10^{-4} \times n = 401$). It is clear from the comparison that ASP outperforms the MRL algorithm, and shows comparable performance to the GK algorithm, even though it is not tuned for the quantiles problem.

Additional experimental results including range queries, etc., are omitted from this extended abstract due to lack of space.

5 Closing Remarks

We have proposed a novel summary for a stream of multidimensional points. This scheme provides a general-purpose summary of the spatial distribution of the points and can answer a wide variety of queries. In particular, ASP can be used to solve spatial analogues of several classical data stream problems, including hot spots and approximate range counting. When specialized to single-dimensional data, adaptive spatial partitioning provides alternative schemes for finding the most frequent items and approximate quantiles, with performance comparable to previous algorithms. Our scheme extends to the sliding window model with a $\log(\varepsilon n)$ factor increase in space.

References

1. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM* **51** (2004), pp. 606–635.
2. N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* **58** (1999), 137–147.
3. A. Arasu and G. Manku. Approximate counts and quantiles over sliding windows. In *Proc. 23rd PODS*, 2004, ACM Press, pp. 286–296.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *21st PODS*, 2002, ACM Press, pp. 1–16.
5. J.L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM* **23** (4) (1980) 214–229.
6. M. Charikar, L. O’Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proc. 35th STOC*, 2003, pp. 30–39.
7. G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *Proc. 29th Conf. VLDB*, 2003.
8. G. Cormode and S. Muthukrishnan. Radial histograms for spatial streams. Technical report DIMACS TR 2003-11, 2003.
9. G. Cormode and S. Muthukrishnan. What is hot and what is not: Tracking most frequent items dynamically. *Proc. 22nd PODS*, 2003, 296–306.
10. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal of Computing* **31** (6) (2002), 1794–1813.
11. E. D. Demaine, A. López-Ortiz, and J.I. Munro. Frequency estimation of internet packet streams with limited space. *Proc. 10th ESA*, LNCS 2461, 2002, pp. 348–360.

12. C. Estan, S. Savage and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proc. SIGCOMM*, ACM Press, 2003, pp. 137–48.
13. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman. Computing iceberg queries efficiently. In *Proc. 24rd Conf. VLDB*, 1998, pp. 299–310.
14. A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the Universe: Dynamic maintenance of quantiles. In *Proc. 28th Conf. on VLDB*, 2002.
15. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. 20th SIGMOD*, 2001, pp. 58–66.
16. J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. In *Proc. 23rd PODS*, 2004, ACM Press, pp. 252–262.
17. R.M. Karp, S. Shenker, and C.H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems* **28** (1) (2003), 51–55.
18. G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. 28th Conf. VLDB*, 2002, pp. 346–357.
19. G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. 17th SIGMOD*, 1998, pp. 426–435.
20. G. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *Proc. 18th SIGMOD*, 1999, pp. 251–262.
21. J. Misra and D. Gries. Finding repeated elements. *Sci. Comput. Programming* **2** (1982), 143–152.
22. J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science* **12** (1980), 315–323.
23. S. Muthukrishnan. Data streams: Algorithms and applications. Preprint, 2003.
24. S. Suri, Cs.D. Tóth, and Y. Zhou, Range counting over multi-dimensional data streams. *Proc. 20th ACM Symp. Comput. Geom.*, ACM Press, 2004, pp. 160–169.
25. N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *Proc. SIGMOD Conf. on Management of Data*, ACM Press, 2002, 428–439.
26. V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.* **16** (1971), 264–280.
27. J.S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Software* **11** (1985), 37–57.

Paired Pointset Traversal

Peter Hui¹ and Marcus Schaefer²

¹ Department of Computer Science, DePaul University, Chicago, Illinois 60604

`phui@students.depaul.edu`

² Department of Computer Science, DePaul University, Chicago, Illinois 60604

`mschaefer@cti.depaul.edu`

Abstract. In the PAIRED POINTSET TRAVERSAL problem we ask if, given two sets $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ in the plane, there is an ordering π of the points such that both $a_{\pi(1)}, \dots, a_{\pi(n)}$ and $b_{\pi(1)}, \dots, b_{\pi(n)}$ are self-avoiding polygonal arcs? We show that PAIRED POINTSET TRAVERSAL is **NP**-complete. This has consequences for the complexity of computing the Fréchet distance of two-dimensional surfaces. We also show that the problem can be solved in polynomial time if the points in A and B are in convex position, and derive some combinatorial estimates on $\text{lct}(A, B)$, the length of a longest common traversal of A and B .

1 Introduction

Suppose we are given two sets of points $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ in the plane. How hard is it to determine whether we can traverse both A and B in the same order such that the resulting polygonal arcs do not self-intersect? Phrased differently, is there a permutation π such that both $a_{\pi(1)}, \dots, a_{\pi(n)}$ and $b_{\pi(1)}, \dots, b_{\pi(n)}$ are self-avoiding polygonal arcs? We call this problem the PAIRED POINTSET TRAVERSAL problem, and call such a permutation a common traversal of A and B . In Section 2 we show that deciding PAIRED POINTSET TRAVERSAL is **NP**-complete.

In Section 3 we study a variant of the PAIRED POINTSET TRAVERSAL problem, in which instead of straight lines we allow curves to connect the points, but restrict the curves to lie within the convex hull of their pointsets. This version can be solved in polynomial time. If we define $\text{lct}(A, B)$ to be the length of a longest common traversal of A and B , we show that in the convex hull case, $\text{lct}(A, B)$ is of order \sqrt{n} .

The PAIRED POINTSET TRAVERSAL problem is of interest in relation to a notion from geometry, the Fréchet distance. Intuitively, the Fréchet distance between two geometric objects is the largest distance between two points in a simultaneous “smooth” traversal of both objects. For example, for one-dimensional curves $f : [0, 1] \rightarrow \mathbb{R}$ and $g : [0, 1] \rightarrow \mathbb{R}$, the Fréchet distance is

$$\delta_F(f, g) = \inf_{\alpha, \beta} \max_{s \in [0, 1]} d(f(\alpha(s)), g(\beta(s))),$$

where α and β range over all continuous and monotone parameterizations of $[0, 1]$. The Fréchet distance captures, better than the more familiar Hausdorff distance does, the notion of similarity in shape.

The Fréchet distance between two polygonal arcs can be found in polynomial time [1]. However, computing the Fréchet distance between two two-dimensional objects (even if restricted to the Euclidean plane) is **NP**-complete [5]. One possible approach to approximating the Fréchet distance would be through sampling: selecting pairs of points from the two objects at random, reordering them so both of them form a polygonal arc, and then computing the Fréchet distance between the two arcs. This procedure is still **NP**-hard, however, since the re-ordering of the pairs of points is the same as finding a common traversal of two pointsets.¹ This connection with the Fréchet distance makes approximation results of interest, and we take some initial steps in that direction in Section 3.2.

2 Paired Pointset Traversal

We formally define our problem **PAIRED POINTSET TRAVERSAL**:

INSTANCE: Two planar pointsets $A = \{a_1, \dots, a_n\}, B = \{b_1, \dots, b_n\}$.

QUESTION: Is there a permutation π such that $(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)})$ and $(b_{\pi(1)}, b_{\pi(2)}, \dots, b_{\pi(n)})$ both form self-avoiding polygonal paths?

Note that we do not prevent the two paths $(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)})$ and $(b_{\pi(1)}, b_{\pi(2)}, \dots, b_{\pi(n)})$ from intersecting — the only restriction that we place upon the paths is that neither path may intersect itself. We refer to such paths as

Theorem 1. **PAIRED POINTSET TRAVERSAL** is **NP**-complete.

We show **NP**-hardness by a reduction from the following version of **HAMILTONIAN PATH**:

INSTANCE: A plane graph $G = (V, E)$ (that is a graph embedded in the plane without intersections), and two of its vertices, s and t .

QUESTION: Does G contain a Hamiltonian path from s to t .

This problem is well-known to be **NP**-complete [4], even if we assume that s and t are on the boundary of the convex hull formed by V (an assumption we will need later).

Given the plane graph $G = (V, E)$ and two vertices s and t , we construct vertex sets $A = \{a_1, a_2, \dots, a_n\}$ and $B = \{b_1, b_2, \dots, b_n\}$ in the plane such that there is Hamiltonian path from s to t in G if and only if A and B allow a matched self-avoiding path. This immediately shows that **PAIRED POINTSET TRAVERSAL** is **NP**-complete.

¹ This connection was pointed out to us by Binhai Zhu, who also supplied us with the problem in the first place [10].

Let $G = (V, E)$ be a plane graph with two special vertices s and t . We start by adding the vertices of V to both A and B . We refer to these vertex sets as V_A and V_B , respectively. Then, for each vertex pair $(u, v) \notin E$, we select a point on the straight line \overline{uv} and add it to B , making sure that the point does not lie on any other straight line \overline{ab} with $a, b \in V$; we call the set of these vertices N_B . These points will assure that we cannot traverse B by going directly from u to v for any edge (u, v) that is absent from E . We then add a corresponding set of vertices, N_A to A . The vertices in N_A have the same relative position to each other as the vertices in N_B , but they appear translated significantly beneath the vertices of V_A so as to allow space for another gadget that will be placed between V_A and N_A . We also add to N_A and N_B another set T_A and T_B , which we will specify later in the proof. T_A and T_B will be necessary to ensure that a simultaneous traversal is possible.

Finally, we need a third set of vertices to prevent traversals of A that include edges going back and forth between V_A to N_A . More precisely, we will add sets I_A and I_B such that any pair of matched self-avoiding paths of A and B will (i) have to start in I_A (I_B) and traverse all of I_A (I_B) with the exception of a single vertex i_A (i_B), then (ii) traverse all of V_A (V_B), (iii) go to i_A (i_B), and finally (iv) traverse all of N_A (of course the traversal could be in the reverse order). We will show how to construct this anti-interference gadget later, and first prove the correctness of the construction. Figure 1 illustrates the construction up to this point.

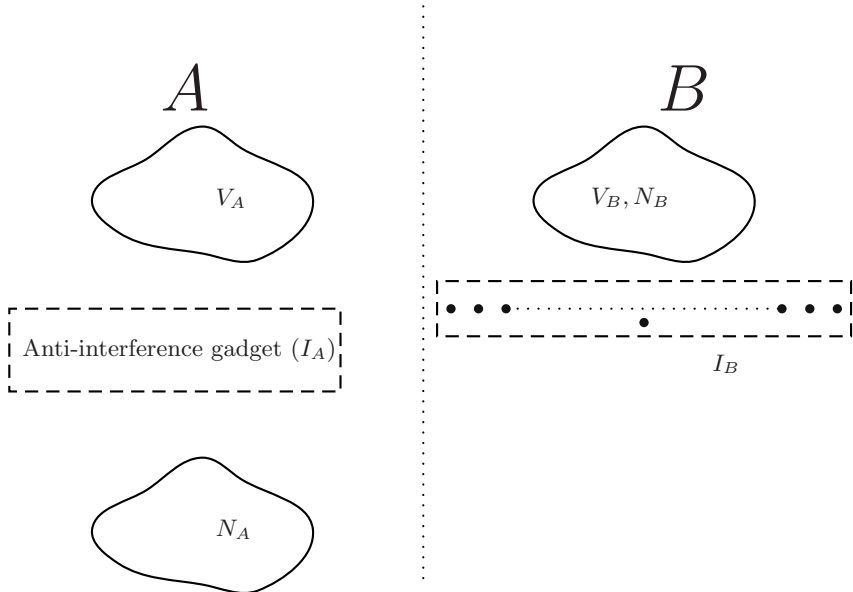


Fig. 1. Reduction from PLANAR HAMILTONIAN PATH

Assuming then that the interference gadget works correctly, and we can choose T_A and T_B to make a traversal of N_A and N_B possible, we can prove the correctness of the construction. First suppose that there is a Hamiltonian

path in G . We can then traverse A and B simultaneously as follows. Begin with $I_A - \{i_A\}$ in A ($I_B - \{i_B\}$ in B). Continue to s , and follow a Hamiltonian path through V_A (V_B) to t . From t go to i_A (i_B), and then on to N_A (N_B). We will show later how to build N_A and N_B such that they can be traversed simultaneously. For the other direction, suppose there is a pair of matched self-avoiding path. Since we assumed that the anti-interference gadget works in that case, we know that the paths traverse all of V_A (V_B) before they begin traversing N_A (N_B). Furthermore, The vertices in N_B ensure that the path through V_B does not use any edges outside of E . Hence, the traversal of V_B constitutes a Hamiltonian path in G , which is what we had to show.

We still owe the reader two details of the construction: how to build the anti-interference gadget, and how to ensure that N_A and N_B can be traversed simultaneously.

Let us first show how to build the anti-interference gadget I_A and I_B . Our goal is to arrange the vertices of I_A and I_B such that all the points in V_A (V_B) get traversed before the points in N_A (N_B), or vice versa. We accomplish this by arranging the vertices of I_A in a series of nested wedges, and the vertices of I_B in a line (see Figure 2).

We claim that there is only one simultaneous traversal of I_A and I_B , namely the one shown in the figure: I_A must be traversed starting at a_1 , continuing outwards forming a series of nested wedges to a_{k+1} . If the claim is true, then the anti-interference gadget performs its function: it separates the traversal of V_A from the traversal of N_A , and, furthermore, its vertices do not interfere with the traversals of V_A and N_A .

We will establish the claim in several steps. We choose $k = 5(|V_B| + |N_B|) + 4$. Consider the vertices along the line b_1, b_2, \dots, b_{k-1} . These vertices can only be traversed in order, unless a traversal uses a vertex outside of that line. Now there are at most $|V_B| + |N_B| + 3$ vertices other than b_1, b_2, \dots, b_{k-1} . This means a traversal must contain four neighboring vertices ($b_i, b_{i+1}, b_{i+2}, b_{i+3}$) (since k is large enough). This, in turn implies that I_A contains a wedge, namely (a_i, a_{i+1}, a_{i+2}) if i is odd, or $(a_{i+1}, a_{i+2}, a_{i+3})$ if i is even. The subsequent vertices, a_{i+3}, \dots, a_{k-2} (or a_{i+4}, \dots, a_{k-2} as the case might be), are entirely shielded from N_A and V_A , which means, since the corresponding b vertices form a line, that they have to be traversed in order, ending with a_{k-2} . The arrangement of the vertices ensures that a_{k-1} is the only point that we can connect to next. Therefore, (b_{k-2}, b_{k-1}) must be present as well. Observe that (b_{k-2}, b_{k-1}) separates b_k from V_B and N_B .

We now show that from a_{k-1} , the only possible next vertex is a_k .

1. We cannot proceed from a_{k-1} to a_{k+1} , since doing so would result in a self-intersection in B .
2. We cannot proceed from a_{k-1} to any vertex in V_A , since doing so would eventually require one of the following next steps:
 - (a) $\dots \dots a_{k+1}$ From a_{k+1} we could then only continue on to V_A , since N_A , and i_A are shielded from it. From V_A we would then have to continue to i_A . At this point, both b_k and the vertices of N_B have yet to

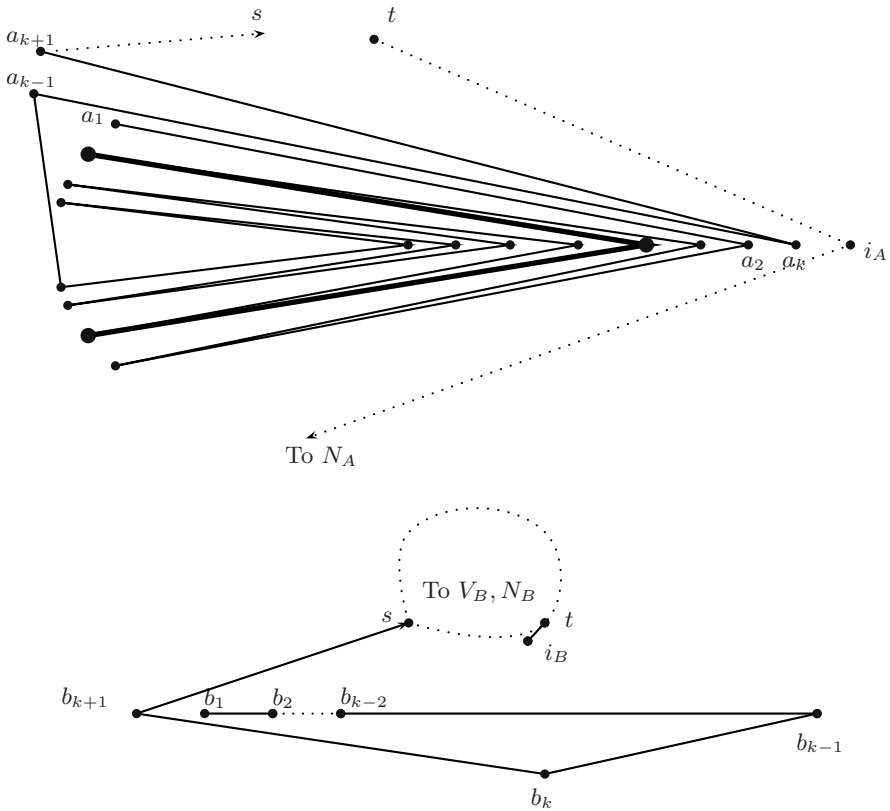


Fig. 2. Anti-interference gadget for pointsets A (top) and B (bottom), along with their sole legal traversal

be visited. We have already proven that we cannot traverse between b_k and N_B , so regardless of whether we decide to visit b_k or N_B , it will be impossible to visit the other.

(b) $\dots \dots i_A$ At this point we can continue to either N_A or $V_A \cup \{a_{k+1}\}$, but we can no longer traverse both.

3. We cannot proceed from a_{k-1} to i_A , since doing so would make it impossible to traverse both V_A and N_A (as above).

This leaves a_k as the sole possible next vertex from a_{k-1} , which in turn implies that the vertices of I_A are traversed as a complete sequence of nested wedges (since (a_{k-1}, a_k) separates V_A from the wedges).

Finally, we have to show how to wire N_A and N_B using additional vertex sets T_A and T_B such that a simultaneous traversal is possible. Remember that N_B contains vertices chosen to obstruct edges that do not occur in E . Now once V_A and V_B (and the anti-interference device) have been traversed, we need to pick out these vertices. To this end, we think of the area of V_B being sepa-

rated into polygonal regions by the complete graph (drawn using straight lines) on the vertices V_B . To each resulting line segment we add four vertices, two on each side of the line segment. Furthermore, if the line segment contains an obstruction point, we make sure it is in the middle between the new vertices. Finally, we add vertices surrounding V_A , so we can pick up any region that is not visible from I_A . Figure 3 illustrates the construction. It shows a particular Hamiltonian path having been chosen (in bold), and, based on that, the subsequent traversal of N_B to pick up all remaining vertices. Since N_A is a translation of N_B , and does not interact with V_A , any legal traversal of N_B is a legal traversal of N_A . Hence, we only need to show that there is a legal traversal of N_B . The Hamiltonian path separates the faces of the complete graph into a number of connected regions, all of which are visible from the outside (a path cannot enclose a region). We can therefore trace a path around the graph, and traverse each connected region we see in a depth-first manner (not entering a face if it has been entered before). In this way we can reach all the points of N_B .

This completes the construction.

□

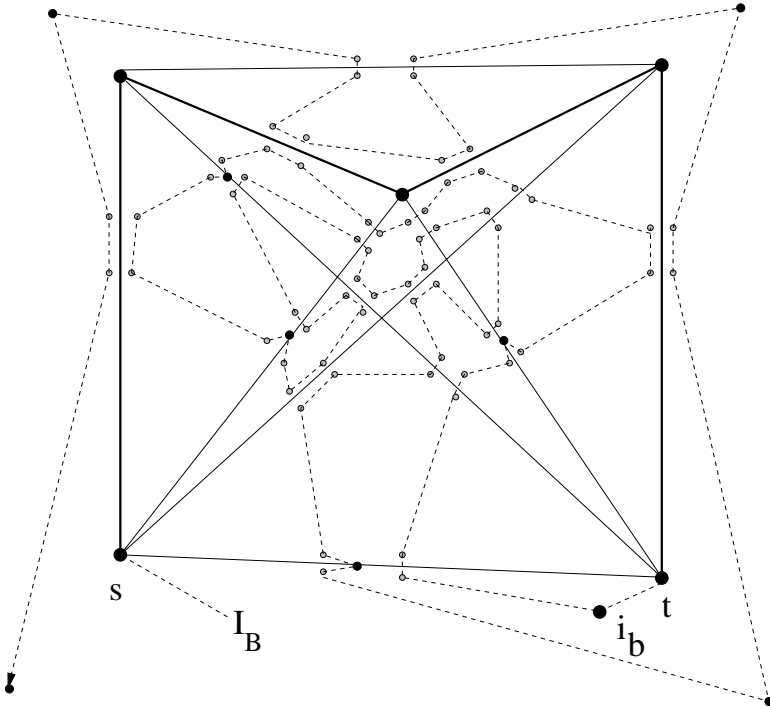


Fig. 3. A graph containing a Hamiltonian path (in bold) with vertices N_B added, along with its traversal (dotted)

3 The Convex Hull Case

Next, we consider a version of the problem in which we allow arbitrary curves (rather than straight lines) between points. This change by itself would make the pointset problem trivial: there always is a solution. However, the situation changes, if we require the curves to be contained within the convex hull formed by their respective pointsets. In that case, a simultaneous traversal is not always possible.

If the pointsets contain points (a_i, b_i) such that either a_i is interior to the convex hull of A , or b_i interior to the convex hull of B (or both), then we can ignore these points, since they do not affect the existence of a paired pointset traversal: at least one of a_i or b_i can be positioned arbitrarily by distorting the interiors. The only problem these points pose is finding them; this can be done in time $O(n \log n)$ using a standard convex hull algorithm.

From this point onwards, we assume that the points in A and B are in convex position; that is, all the points of A and B lie on the boundary of their convex hulls.

While the initial problem is NP-complete, we show that the convex hull variant can be solved in linear time using a combination of greedy choice and dynamic programming.

3.1 Recognition

Let us first concentrate on legal traversals within a single set in convex position. For example, Figure 4 shows a legal traversal of a set in convex position.

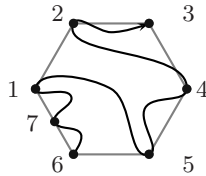


Fig. 4. A pointset with a legal traversal, $(6, 7, 1, 5, 4, 2, 3)$. We allow arbitrary curves between points, provided that all curves lie within the convex hull of their pointsets

Let (v_1, v_2, \dots, v_n) be a clockwise ordering of the vertices of A on the border of the convex hull. We claim that any legal traversal of all the vertices in A which starts with v_1 can be obtained as follows: beginning with the sequence (v_2, \dots, v_n) keep removing either the first or the last vertex from the sequence and add it to the traversal.

For example, we start with the clockwise ordering $(6, 7, 1, 2, 3, 4, 5)$ in Figure 4, and continue $(7, 1, 2, 3, 4, 5)$, $(1, 2, 3, 4, 5)$, $(2, 3, 4, 5)$, $(2, 3, 4)$, $(2, 3)$, (3) , $(\)$, obtaining the legal traversal $(6, 7, 1, 5, 4, 2, 3)$.

The proof of the claim is easy: if we begin with v_1 and then select a vertex v_i other than v_2 or v_n , we disconnect v_2, \dots, v_{i-1} from v_{i+1}, \dots, v_n (both non-empty sets). On the other hand, any traversal obtained in the fashion described leads to a legal traversal, that is, a self-avoiding path.

The situation becomes more involved if instead of a traversal of one set, we consider simultaneous traversals of two sets, A and B . If we assume, for the moment, that we know the starting points a_1 and b_1 of the traversals in both A and B , then we know that in each set there are at most two ways of continuing a self-avoiding path. There are three possibilities: if there is no common way to continue the traversals, we have run into a dead end. If there is exactly one common way to continue the traversal, we choose that continuation. The only problem arises if both ways of continuing the current traversal work are legal in both A and B . The following lemma shows that in that case it does not matter which choice we make: if we have a choice between two vertices to select, then choosing one of the vertices will enable us to complete a pair of self-avoiding paths if and only if choosing the other will as well.

Lemma 1. Let $A = (a_1, a_2, a_3, \dots, a_n)$ and $B = (b_1, b_{\pi(2)}, b_{\pi(3)}, \dots, b_{\pi(n)})$ be two disjoint pointsets. Let π be a permutation of $\{2, \dots, n\}$. Then the following two conditions are equivalent: (i) $\pi(2) = 2, \dots, \pi(n) = n$; (ii) $\pi(2) = n, \dots, \pi(n) = 2$. In both cases, the traversals starting at (a_1, a_2) and (b_1, b_2) can be extended to (a_n, a_n) and (b_n, b_n) .

We only consider the case $\pi(2) = 2$ (the other being symmetric: take a mirror image of one of A or B). If there is a simultaneous traversal, we can without loss of generality assume that there is one starting with a_1, a_2 and, therefore, b_1, b_2 . Follow the traversal, until it chooses a_n, b_n ; that is, we pick $a_1, a_2, \dots, a_i, a_n$ and $b_1, b_2, \dots, b_i, b_n$. But then we might as well pick $a_1, a_n, a_2, \dots, a_i$ and $b_1, b_n, b_2, \dots, b_i$. In both cases we are left with the same set of remaining vertices: $(a_{i+1}, \dots, a_{n-1})$ and $(b_{i+1}, \dots, b_{n-1})$. Hence, if we have a choice of how to continue after picking a_1 (b_1) it does not matter which choice we select. □

At this point, we can write an algorithm which, given index i and length l , determines in linear time whether or not a paired self-avoiding path fragment of length l exists traversing A and B and starting at a_i and b_i (see Algorithm A in the appendix). We could then extend this algorithm by a brute force approach and try each $(a_i, b_i), 1 \leq i \leq n$, to determine whether a simultaneous traversal exists in quadratic time. However, with a slight extension of this algorithm, we can tighten this bound to make the entire algorithm run in linear time. We do this by using dynamic programming. We build segments of a path starting at an arbitrary (a_i, b_i) , saving the computed path for subsequent attempts at a different (a_j, b_j) in the event that this one fails. In this manner, no subpath will ever need to be recomputed. For lack of space we do not include details.

3.2 Optimization

If Algorithm A determines that A and B do not have a common pointset traversal, we could still try to optimize the length of a longest common traversal, $\text{lct}(A, B)$. We suspect that this problem will turn out to be **NP**-complete, hence approximation results would be of interest. A simple approximation result follows from the following observation:

Lemma 2. A and B are two sets of n vertices in convex position. $\text{lct}(A, B) \geq \sqrt{n}$

Assume the vertices are labelled clockwise as a_1, \dots, a_n , and the vertices of b as $b_{\pi(1)}, \dots, b_{\pi(n)}$ for some permutation π of $\{1, \dots, n\}$. Then a longest monotone subsequence of $(\pi(1), \dots, \pi(n))$ will be a common legal traversal of A and B . An old result of Erdős and Szekeres shows that any sequence of distinct real numbers contains a monotone subsequence of length at least \sqrt{n} [2, 9]. \square

The lemma allows us to approximate $\text{lct}(A, B)$ to within a factor of $n^{1/4}$ by simply predicting it to be $n^{3/4}$. We also note that the lemma is asymptotically optimal:

Lemma 3. A and B are two sets of n vertices in convex position. $\text{lct}(A, B) \leq 2\sqrt{n} + 1$

Consider two sets A and B , each in convex position, and a common (not necessarily full) traversal. Without loss of generality we can assume that the vertices of A are labelled a_1, \dots, a_n in a clockwise ordering, and the common traversal begins with a_1 and b_1 . If we consider the sequence of indexes of this traversal it has the property that it alternately increases and decreases, without ever going beyond any earlier values. That is if $i < j < k$, then either a_i, a_j, a_k is monotone, or a_k lies between a_i and a_j . We call such a sequence a telescope sequence. For example, 1, 3, 13, 11, 9, 5, 8, 7 is a telescope sequence. If we split a telescope sequence into its increasing and its decreasing parts, it is clear that every telescope sequence of length m contains a monotone sequence of length at least $m/2$. Returning to our traversal, we see that if the traversal has length m , then it contains a monotone traversal of length at least $m/2$ whose indices form a monotone sequence. Hence, if we order B in such a way that it does not have a monotone subsequence of length $\sqrt{n} + 1$ (which is possible, since the result by Erdős and Szekeres is sharp), then $\text{lct}(A, B) \leq 2\sqrt{n} + 1$. \square

The convex case gives us a first break into the general case.

Corollary 1. A and B are two sets of n vertices in convex position. $\text{lct}(A, B) \geq n^{1/18}$

By a result of Raynaud [8–Theorem 4.2] the convex hull of A has an expected number of $n^{1/3}$ vertices on its boundary. If we restrict B to the corresponding vertices, those $n^{1/3}$ vertices are still uniformly distributed over the circle, hence their convex hull has an expected number of $n^{1/9}$ vertices on the boundary. If we restrict A to the corresponding vertices, we are now in a situation where both A and B are in convex position, and we can apply Lemma 2 to conclude that we expect $\text{lct}(A, B)$ to be at least $n^{1/18}$. \square

4 Open Questions

As far as we know this is the first time the common pointset traversal problem has been formalized and studied, so open questions still abound, even though the convex hull case seems to be closely tied to results from geometric Ramsey theory.

In the general case, the most important open problem is the complexity of approximating $\text{lct}(A, B)$. The construction in our **NP**-hardness result is very fragile, and it seems tricky to even extend it to claiming that approximating $\text{lct}(A, B)$ to within an additive constant is still **NP**-hard.

We did show that $\text{lct}(A, B)$ is at least $n^{1/18}$ for points chosen at random in a circle; the result relied on finding large subsets of points in convex positions. By a well-known result of Erdős and Szekeres [3, 7] there are pointsets such that the largest subset of points in convex position is of order $\log n$. Hence using the same idea as in the random case, the best result we can obtain is that $\text{lct}(A, B) \geq \sqrt{\log \log n}$, which cannot possibly be anywhere near the truth.

In the convex hull case, the most immediate problem is to settle the complexity of the optimization problem. However, there are several other questions that also suggest themselves: how fast can we solve the paired pointset traversal problem if we allow up to k intersections. Is the resulting problem **NP**-complete? If so (and we conjecture that this is the case), how does the complexity vary with k ? More precisely, taking the parameterized complexity view, can we show that the paired pointset traversal problem can be solved in time $O(n^c)$ for any fixed k , where c does not depend on k . A recent, comparable result, of Martin Grohe shows that we can determine in quadratic time, whether the crossing number of a graph is at most k (for any fixed k) [6].

Acknowledgment. We would like to thank Binhai Zhu for suggesting the problem to us, and explaining its application to measuring similarity of two-dimensional surfaces.

References

1. Helmut Alt and Michael Godau. Measuring the resemblance of polygonal curves. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 102–109. ACM Press, 1992.
2. Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.
3. Paul Erdős and George Szekeres. On some extremum problems in elementary geometry. *Ann. Univ. Sci. Budapest Eotvos Soc. Math.*, 3-4:53–62, 1961.
4. M. R. Garey, D. S. Johnson, and R. Endre Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, December 1976.
5. Michael Godeau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. PhD thesis, Freie Universität Berlin, 1998.
6. Martin Grohe. Computing crossing numbers in quadratic time. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 231–236, 2001.

7. W. Morris and V. Soltan. The Erdős-Szekeres problem on points in convex position—a survey. *Bull. Amer. Math. Soc. (N.S.)*, 37(4):437–458, 2000.
8. Franco P. Preparata and Michael Ian Shamos. *Computational geometry. An introduction*. Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985.
9. J. Michael Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In *Discrete probability and algorithms (Minneapolis, MN, 1993)*, volume 72 of *IMA Vol. Math. Appl.*, pages 111–131. Springer, New York, 1995.
10. Binhai Zhu, 2004. Personal Communication.

A Convex Hull Algorithm (Given Starting Points)

Algorithm 1 Linear-time algorithm for deciding if a matched self-avoiding path of length l through A and B exists starting at points a_i and b_i

```

1:  $PATH \leftarrow \lambda$ 
2:  $C_A \leftarrow \text{ClockwiseOrdering}(A, a_i)$ 
3:  $C_B \leftarrow \text{ClockwiseOrdering}(B, b_i)$ 
4: while ( $|PATH| < l$ ) do
5:   if ( $(C_A[\text{first}] = C_B[\text{first}])$  OR ( $C_A[\text{first}] = C_B[\text{last}]$ )) then
6:      $newVertex \leftarrow C_A[\text{first}]$ 
7:     add  $newVertex$  to  $PATH$ 
8:     delete  $newVertex$  from  $C_A$ 
9:     delete  $newVertex$  from  $C_B$ 
10:  else if ( $(C_A[\text{last}] = C_B[\text{first}])$  OR ( $C_A[\text{last}] = C_B[\text{last}]$ )) then
11:     $newVertex \leftarrow C_A[\text{last}]$ 
12:    add  $newVertex$  to  $PATH$ 
13:    delete  $newVertex$  from  $C_A$ 
14:    delete  $newVertex$  from  $C_B$ 
15:  else
16:    return false    cannot proceed any farther
17:  end if
18: end while
19: return true    both orderings have been consumed

```

Approximated Two Choices in Randomized Load Balancing

Kazuo Iwama* and Akinori Kawachi**

Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
{iwama, kawachi}@kuis.kyoto-u.ac.jp

Abstract. This paper studies the maximum load in the *approximated* d -choice balls-and-bins game where the current load of each bin is available only approximately. In the model of this game, we have r thresholds T_1, \dots, T_r ($0 < T_1 < \dots < T_r$) for an integer r (≥ 1). For each ball, we select d bins and put the ball into the bin of the lowest range, i.e., the bin of load i such that $T_k \leq i \leq T_{k+1} - 1$ and no other selected bin has height less than T_k . If there are two or more bins in the lowest range (i.e., their height is between T_k and $T_{k+1} - 1$), then we assume that those bins cannot be distinguished and so one of them is selected uniformly at random. We then estimate the maximum load for n balls and n bins in this game. In particular, when we put the r thresholds at a regular interval of an appropriate Δ , i.e., $T_r - T_{r-1} = \dots = T_2 - T_1 = T_1 = \Delta$, the maximum load $L(r)$ is given as $(r + O(1))^{r+1} \sqrt{\frac{r+1}{(d-1)^r}} \ln n / \ln \left(\frac{r+1}{(d-1)^r} \ln n \right)$. The bound is also described as $L(\Delta) \leq \{(1 + o(1)) \ln \ln n + O(1)\} \Delta / \ln((d-1)\Delta)$ using parameter Δ . Thus, if Δ is a constant, this bound matches the (tight) bound in the original d -choice model given by Azar et al., within a constant factor. The bound is also tight within a constant factor when $r = 1$.

1 Introduction

Suppose that we put n balls into n bins by selecting a bin uniformly at random for each ball. This simple model, often called the *balls-and-bins* model, has been quite popular for studying several properties in both mathematics and computer-science fields (see, e.g., [14]). The best known property in this model is that the load of the highest bin, or the maximum number of balls in any bin, is approximately $\ln n / \ln \ln n$ with high probability when the game is finished. In [2], Azar, Broder, Karlin and Upfal showed that the situation changes dramatically if we are allowed to select two bins instead of one for each ball: Suppose that two bins are selected uniformly at random and the ball is put into the lower bin of the two. Then, the maximum load is approximately $\ln \ln n$ with high

* Supported in part by Scientific Research Grant, Ministry of Japan, No. 13480081, No. 15630001, No. 16300003, and No. 16092101.

** Supported in part by the 21st Century COE Program (Grant No. 14213201).

probability, which is exponentially smaller than the one-choice case. (This result also stimulated the research and was followed by several other results [1, 15, 4, 9, 5, 10].

This remarkable improvement is obviously due to the fact that we can select two bins instead of one, but the following fact should be equally important as well: Namely, we can tell which of the two selected bins is lower than the other \dots , even if the difference of the height is only one. In this paper, we study the case where we can tell the difference of the height of the two bins \dots , i.e., only if the difference is larger than some value.

There are two reasons why we are interested in such a model which we call \dots , or more in general $\dots d \dots$. The first reason is related to its application: Among others, the most natural application of the balls-and-bins game is load balancing. Suppose that a sequence of tasks is assigned to servers. Then the one-choice model corresponds to selecting one server at random for each task and the two-choice model to selecting two servers and assigning the task to the server which is less loaded than the other. By using the latter, we can obtain an exponentially better load balance among the servers. To make this possible, however, we have to know the \dots loading-factor of each server at each moment, which does not seem easy in some situations. The second reason is more theoretical: The large gap in the maximum load between the two models naturally leads us to consider an intermediate model which bridges the two models rather continuously. This approach is popular and usually gives us useful insights on why those two models are so different. (For instance, see [13] for the difference in the satisfiability threshold between random 2SAT and random 3SAT.)

Our Contribution. Suppose that we can tell which bin is less loaded than the other iff the number of balls currently loaded in the two bins differs by at least Δ . Then our main result shows that the maximum load heavily depends on this value Δ . Actually in this paper, we analyze the more general d -choice model and for technical reason we use the following “threshold model.” Let r be an integer (≥ 1) and T_1, \dots, T_r be r different integers ($0 < T_1 < T_2 < \dots < T_r$) called \dots . For each ball, we select d bins and put the ball into the bin of the lowest range, i.e., the bin of load i such that $T_k \leq i \leq T_{k+1} - 1$ and no other selected bin has height less than T_k . If there are two or more bins in the lowest range (i.e., their height is between T_k and $T_{k+1} - 1$), one of them is selected uniformly at random. (See Fig. 1.) Then the maximum load $L(r)$ is given as

$$L(r) \leq (r + O(1))^{r+1} \sqrt{\frac{\frac{r+1}{(d-1)^r} \ln n}{\ln \left(\frac{r+1}{(d-1)^r} \ln n \right)}}$$

The bound is also described as

$$L(\Delta) \leq \{(1 + o(1)) \ln \ln n + O(1)\} \frac{\Delta}{\ln((d-1)\Delta)}$$

using parameter Δ . Thus, if Δ is a constant, this bound matches the (tight) bound in the original d -choice model given by Azar et al., within a constant factor. The bound is also tight within a constant factor when $r = 1$.

Thus we can obtain a rather continuous change depending on the simple parameter r , which was our main target of the research. Some remarks are as follows: (1) Although our main theorem is more general, the above particular bounds (both for $L(r)$ and $L(\Delta)$) are obtained when the thresholds are set regularly, namely such that $T_1 = T_2 - T_1 = \dots = T_r - T_{r-1} = \Delta$. Note that we can prove that this is the way of getting the best load balance when $d = 2$ and it is probably true also for a general d . Under this regularity assumption, there must be at least one threshold between the heights h_1 and h_2 of the two bins if $|h_1 - h_2| \geq 2\Delta$, and so the above bound for $L(\Delta)$ immediately follows. (2) By increasing d , we can obtain a better load balance. This merit is given as (i) a factor of $1/\sqrt{d}$ when $d = 2$, (ii) a factor of $1/d^{r/r+1}$ when r is a constant, and (iii) the factor approaches to $1/\ln d$ when r is large. (Details are omitted but it is not hard to calculate these factors from the bound for $L(r)$.) Thus the merit of increasing d reaches a maximum for some value of r . (See below for other analysis of this merit.) (3) This upper bound for $L(r)$ is tight within a constant factor when $r = 1$. Our strong conjecture is that it is also similarly tight for a general r , but the lower-bound analysis appears to get much more complicated.

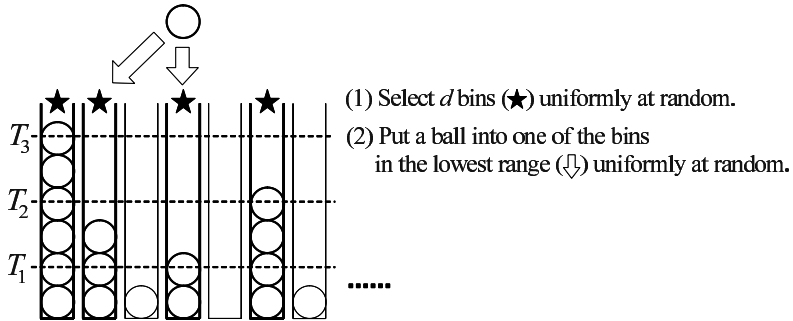


Fig. 1. Approximated d -choice balls-and-bins game

Related Results. The first analytical results on the power of multiple choices were demonstrated by Karp, Luby and Meyer auf de Heide [6] in the context of the PRAM simulations. As mentioned before, [2] is a seminal paper which showed that the maximum load in the d -choice ($d \geq 2$) is $\frac{\ln \ln n}{\ln d} + \Theta(1)$ with high probability. Furthermore, Vöcking showed an improvement of the maximum load in the d -choice model using asymmetric allocations [15, 16]: If we select two or more bins with the same height by d -choice, we always put a ball into the left-most bin among them in his modified model. Then, such a slight modification of the d -choice model yields a maximum load of approximately $\frac{\ln \ln n}{d}$ with high probability, an exponential improvement in terms of d . Berenbink, Czumaj, Steger and Vöcking also investigated the power of multiple choices and asymmetric

allocations with n bins and m balls when $m \gg n$, i.e., the balls-and-bins game in the heavily loaded case [4].

Many settings of the balls-and-bins game have been also considered besides the sequential multiple-choice models. Adler, Chakrabarti, Mitzenmacher and Rasmussen, for example, explored parallel and distributed settings [1] (we can also find their results in [8]). In their settings, each ball must simultaneously decide its destination through communication with other balls. Thus the model (and the analysis as well) is quite different from the sequential model, but interestingly, their results look somehow similar to ours: They proved that the lower bounds of the maximum load is $\Omega\left(r^{+1}\sqrt{\ln n/\ln \ln n}\right)$ through r rounds of communication for a wide class of parallel models, and also gave a constant upper bound of the maximum load with $\ln \ln n/\ln d + O(d)$ rounds of communication for the d -choice model.

There are plenty of other settings such as the feedback model [5], and the memory model [10]. The paradigm of the multiple choices has been applied to several dynamic models, and they succeeded in achieving improvements like ones in the standard setting. [2] actually analyzed a dynamic model with multiple choices in the sequential balls-and-bins game. Mitzenmacher [8, 7, 9] and Vvendenskaya, Dobrushin and Karpelevich [17] proposed queuing theoretic models with multiple choices and gave the analysis of the power of multiple choices in the queuing theory. Mitzenmacher and Vöcking [12] also showed the improved queuing theoretic model by introducing asymmetric allocations.

2 Notations and Useful Lemmas

Most notations in this paper are similar to [2, 11]: $B(n, p)$ is the binomial distribution with parameters n and p . $h(t)$ denotes the random variable of the height of the t -th ball, i.e., “ $h(t) = k$ with probability p ” means that the t -th ball is put into a bin of height $k - 1$ with probability p . $\mu_i^A(t)$ and $\nu_i^A(t)$ denote the number of the balls whose heights are at least i and the number of the bins whose heights are at least i , respectively, at time t under model A . In particular, we define μ_i^A and ν_i^A as $\mu_i^A(n)$ and $\nu_i^A(n)$. We omit the name of the model A in $\mu_i^A(t)$ and $\nu_i^A(t)$ if it is clear from the context. Let AdC be our Approximated d -Choice model and $1C(m, n)$ be the conventional one-choice model with m balls and n bins.

The following three lemmas will be used in the rest of the paper. The first lemma is the Chernoff-type bound, also given in [2].

Lemma 1. X_i ($1 \leq i \leq n$) \dots
 $\dots \Pr[X_i = 1] = p, \dots$

$$\Pr\left[\sum_{i=1}^n X_i \geq np\right] \leq \exp(-np), \quad \Pr\left[\sum_{i=1}^n X_i \leq np/e\right] \leq \exp((2/e - 1)np).$$

The next lemma [1, 8] is convenient when we approximate $1C(m, n)$ by the Poisson distribution.

Lemma 2. Let $\mathcal{E}(X_1, \dots, X_n)$ be the event that $\text{AdC}(m, n)$ fails, where X_i ($i = 1, \dots, n$) are independent Poisson random variables with mean m/n , and Y_1, \dots, Y_n are independent Poisson random variables with mean m/n .

$$\Pr[\mathcal{E}(X_1, \dots, X_n)] \leq 4 \Pr[\mathcal{E}(Y_1, \dots, Y_n)]$$

$$\Pr[\mathcal{E}(X_1, \dots, X_n)] \leq m$$

It is well known that the following bounds hold for tails of the Poisson distribution (e.g., [3]).

Lemma 3. For $m \geq \lambda$, $m + 1 > \lambda$,

$$\frac{\lambda^m e^{-\lambda}}{m!} \leq \sum_{k=m}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \leq \frac{m + 1}{m + 1 - \lambda} \frac{\lambda^m e^{-\lambda}}{m!}.$$

3 Main Results

The following theorem is a general form on the upper bound of the maximum load by AdC .

Theorem 1. Let $S_r = \prod_{l=1}^{r-1} ((d-1)(T_{l+1} - T_l) + 1)$, where $d \geq 2$, $T_1 > C$, and AdC is the d -choice algorithm with thresholds T_1, \dots, T_r .

$$T_r + \frac{\ln n - 2 \ln \ln n}{(d-1)S_r T_1 (\ln T_1 - O(1))} + O(1)$$

$$1 - o(1), \quad S_r T_1 = o(\ln n / \ln \ln n), \quad S_r = \omega(\ln \ln \ln n)$$

The conditions that $S_r T_1 = o(\ln n / \ln \ln n)$ and $S_r = \omega(\ln \ln \ln n)$ are always met in the following argument. The proof is given in the following Sections 3.1–3.4. Several corollaries follow: Corollary 1 is for the case that we set the thresholds regularly, Corollary 2 if we use parameter Δ (= the size of each range) instead of r , and Corollary 3 for the case that Δ is a constant, which is similar to the original d -choice model. We omit the proofs of these corollaries.

Corollary 1. Let $T_k = k\Delta = k \sqrt[r+1]{\frac{r+1}{(d-1)^r} \ln n / \ln \left(\frac{r+1}{(d-1)^r} \ln n \right)}$, where $1 \leq k \leq r$, $\Delta > C$, and AdC is the d -choice algorithm with thresholds T_1, \dots, T_r .

$$(r + O(1)) \Delta = (r + O(1)) \left(\frac{\frac{r+1}{(d-1)^r} \ln n}{\ln \left(\frac{r+1}{(d-1)^r} \ln n \right)} \right)^{\frac{1}{r+1}}$$

$$1 - o(1), \quad d \geq 2$$

Corollary 2. $\dots T_k = k\Delta = k \sqrt[r+1]{\frac{r+1}{(d-1)^r} \ln n / \ln \left(\frac{r+1}{(d-1)^r} \ln n \right)} \dots 1 \leq k \leq r \dots \Delta > C \dots C, \dots L(\Delta) \dots AdC \dots$

$$\{(1 + o(1)) \ln \ln n + O(1)\} \frac{\Delta}{\ln((d-1)\Delta)}.$$

$\dots 1 - o(1) \dots d \dots$

Corollary 3. $\dots \Delta = C \dots C, \dots AdC \dots (1 + o(1))C \frac{\ln \ln n}{\ln(C(d-1))} + O(1) \dots 1 - o(1) \dots d \dots$

3.1 Proof of Theorem 1

Our main tool for the proof is the so-called $\dots [2]$. The layered induction is based on the following idea. Let p_i be the maximum probability that we put a ball into one of the bins whose heights are i or more. Obviously $\nu_{i+1}(n)$ is bounded above by $cn \cdot p_i$ with high probability for some constant c . Note that the probability p_i depends on the value of $\nu_i(n)$. We therefore construct a recurrence $\{\alpha_i\}_i$ that bounds $\nu_i(n)$ above for every i from the relation $\alpha_{i+1} = cn p_i$ where p_i depends on α_i . We can thus find the maximum load by estimating i such that $\alpha_i < 1$.

Our proof consists of three stages. In the first stage, we bound ν_{T_1} , i.e., the number of bins whose height is at least the value of the first threshold. For this purpose, we cannot use the layered induction since the probability p_i for $i < T_1$ is too large to use the technique. (As mentioned above, we use the recurrence $\alpha_{i+1} = cn p_i$, which is obviously not too tight since the total number $cn \cdot r$ of balls is (much) larger than n . Thus the recurrence can be used only when p_i is sufficiently small.) Instead we use the $1C(dn, n)$ model, i.e., we put dn balls into n bins uniformly at random and evaluate the number $\nu_{T_1}^{1C(dn, n)}$ of bins whose height is at least T_1 under this model. Note that dn bins are selected in total uniformly at random in both AdC and $1C(dn, n)$. So suppose that s_1, s_2, \dots, s_{dn} are a sequence of selected bins for both AdC and $1C(dn, n)$. Then the number of balls received by an arbitrary bin under AdC is obviously at most the number of balls received by the bin under $1C(dn, n)$. Thus we can conclude that $\nu_{T_1}^{AdC} \leq \nu_{T_1}^{1C(dn, n)}$.

The second stage is a main part of the layered induction. Note that it is more like a double induction because of the existence of thresholds: Defining an appropriate sequence $\{\alpha_i\}_{T_k \leq i \leq T_{k+1}}$ for any k , we first prove that $\nu_i < \alpha_i$ with high probability for the height i between T_k and T_{k+1} . Next, we prove that $\nu_{T_k} < \alpha_{T_k}$ for a sequence $\{\alpha_i\}_{i=T_1, T_2, \dots, T_r}$ with high probability. Finally, in the third stage, we consider the number of the bins whose heights are more than the last threshold T_r . The proof in the third stage can be done by almost the same argument as [2].

3.2 First Stage

We first prove the following lemma.

Lemma 4.

$$\Pr \left[\nu_{T_1}^{AdC} \geq en \left(\frac{ed}{T_1} \right)^{T_1} \right] \leq 4 \exp \left(-\frac{1}{e^d} \left(\frac{d}{T_1} \right)^{T_1} n \right).$$

As described in Sec 3.1, *AdC* can be replaced by *1C*(*dn*, *n*) in this stage. More formally, we can claim that

$$\Pr \left[\nu_{T_1}^{AdC} \geq en \left(\frac{ed}{T_1} \right)^{T_1} \right] \leq \Pr \left[\nu_{T_1}^{1C(dn,n)} \geq en \left(\frac{ed}{T_1} \right)^{T_1} \right]$$

for any $T_1 \in \{1, \dots, n\}$. Now it suffices to show an upper bound of the right-side probability by the Poisson approximation. For a random variable X , let

$$p_{T_1} = \Pr_{X \in (0, \infty)} [X \geq T_1 : X \text{ has the Poisson distribution with mean } d].$$

Then we have

$$\Pr[\nu_{T_1}^{1C(dn,n)} \geq en p_{T_1}] \leq 4 \Pr[B(n, p_{T_1}) \geq en p_{T_1}] \leq 4 \exp(-n p_{T_1})$$

by Lemmas 2 and 1. Also,

$$\frac{1}{e^d} \left(\frac{d}{T_1} \right)^{T_1} \leq p_{T_1} \leq \left(\frac{ed}{T_1} \right)^{T_1}$$

by Lemma 3 if d is a constant and T_1 is larger than some constant. Therefore,

$$\Pr \left[\nu_{T_1}^{1C(dn,n)} \geq en \left(\frac{ed}{T_1} \right)^{T_1} \right] \leq 4 \exp \left(-\frac{1}{e^d} \left(\frac{d}{T_1} \right)^{T_1} \right).$$

□

3.3 Second Stage

For $i \geq T_1$, we define a sequence $\{\alpha_i\}_{i \geq T_1}$ as follows:

$$\alpha_{T_1} = en \left(\frac{ed}{T_1} \right)^{T_1}, \quad \alpha_{i+1} = \begin{cases} \frac{c_1 e \alpha_{T_k}^{d-1} \alpha_i}{n^d} & \text{for } T_k \leq i < T_{k+1}, k = 1, \dots, r, \\ \frac{c_2 e \alpha_{T_r}^{d-1} \alpha_i}{n^d} & \text{for } i \geq T_r. \end{cases}$$

Then by a simple calculation, we have

$$\alpha_i = \begin{cases} \left(\frac{c_1 e}{n} \right)^{i-T_k} \alpha_{T_k}^{i-T_k+1} & \text{for } T_k \leq i < T_{k+1}, \\ \left(\frac{c_2 e}{n} \right)^{i-T_r} \alpha_{T_r}^{i-T_r+1} & \text{for } i \geq T_r. \end{cases}$$

When we let $c'_1 = (c_1 e)^{1/(d-1)}$, we have

$$\begin{aligned} \alpha_{T_k} &= \left(\frac{c'_1}{n}\right)^{(d-1)(T_k - T_{k-1})} \alpha_{T_{k-1}}^{(d-1)(T_k - T_{k-1} + \frac{1}{d-1})}, \\ &= \left(\frac{c'_1}{n}\right)^{(d-1)^{k-1} \prod_{l=1}^{k-1} (T_{l+1} - T_l + \frac{1}{d-1}) - 1} \alpha_{T_1}^{(d-1)^{k-1} \prod_{l=1}^{k-1} (T_{l+1} - T_l + \frac{1}{d-1})}. \end{aligned}$$

Also, let

$$S_k = \begin{cases} 1 & \text{for } T_1 \leq i < T_2, \\ (d-1)^{k-1} \prod_{l=1}^{k-1} \left(T_{l+1} - T_l + \frac{1}{d-1}\right) & \text{for } T_k \leq i < T_{k+1}, k = 2, \dots, r-1. \end{cases}$$

Then, we can represent α_{T_k} as follows:

$$\alpha_{T_k} = \left(\frac{c'_1}{n}\right)^{S_k - 1} \alpha_{T_1}^{S_k}.$$

Now we can prove that ν_i is bounded above by α_i with high probability.

Lemma 5.

$$\Pr[\nu_{T_r}^{AdC} \geq \alpha_{T_r}] \leq (T_r - T_1) \exp\left(-\frac{c_1}{c_1^d} n \left(c_1 e \left(\frac{ed}{T_1}\right)^{T_1}\right)^{S_r}\right) + \Pr[\nu_{T_1}^{AdC} \geq \alpha_{T_1}].$$

We first consider $\Pr[h(t) > i]$, i.e., the probability that we put the t -th ball into the bins whose heights are i or more. Suppose that d is a constant and $T_k \leq i \leq T_{k+1} - 1$ for some k . Recall that d bins are selected at random for the t -th ball. If a bin whose height is less than T_k is selected, then this probability is obviously zero. In the following, we assume that out of d bins, y bins are of height between T_k and $i - 1$, x bins of height between i and $T_{k+1} - 1$, and $d - x - y$ bins of height T_{k+1} or more (see Fig. 2).

$$\begin{aligned} \Pr[h(t) > i] &= \left(\frac{\nu_i(t-1)}{n}\right)^d + \sum_{x \geq 1, y \geq 0, x+y \leq d} \frac{d!}{x!y!(d-x-y)!} \frac{x}{x+y} \times \\ &\quad \left(\frac{\nu_{T_{k+1}}(t-1)}{n}\right)^{d-x-y} \left(\frac{\nu_i(t-1) - \nu_{T_{k+1}}(t-1)}{n}\right)^x \left(\frac{\nu_{T_k}(t-1) - \nu_i(t-1)}{n}\right)^y \\ &\leq \left(\frac{\nu_i(t-1)}{n}\right)^d + \sum_{x=1}^d \sum_{y=0}^{d-x} \frac{d!}{x!y!(d-x-y)!} \frac{x}{x+y} \left(\frac{\nu_i(t-1)}{n}\right)^{d-y} \left(\frac{\nu_{T_k}(t-1)}{n}\right)^y \\ &\leq \left(\frac{\nu_i(t-1)}{n}\right)^d + \left(\sum_{x=1}^d \sum_{y=0}^{d-x} \frac{d!}{x!y!(d-x-y)!} \frac{x}{x+y}\right) \left(\frac{\nu_i(t-1)}{n}\right) \left(\frac{\nu_{T_k}(t-1)}{n}\right)^{d-1} \\ &\leq c_1 \frac{\nu_i(t-1)\nu_{T_k}(t-1)^{d-1}}{n^d} \tag{1} \end{aligned}$$

for some constant c_1 . Since we can prove this lemma by a similar argument to [2] with the above estimation of $\Pr[h(t) > i]$, we omit the rest of this proof. \square

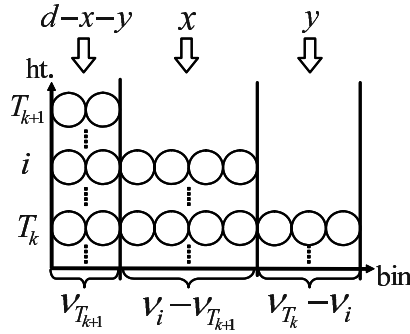


Fig. 2. The estimation of $\Pr[h(t) > i]$

3.4 Third Stage

For $i > T_r$, we can imply the following inequality just as (1) for some constant c_2 . (Recall that we are assuming that d is a constant.)

$$\begin{aligned} \Pr[h(t) > i] &= \sum_{x=1}^d \binom{d}{x} \frac{x}{d} \left(\frac{\nu_i(t-1)}{n}\right)^x \left(\frac{\nu_{T_r}(t-1) - \nu_i(t-1)}{n}\right)^{d-x} \\ &\leq c_2 \frac{\nu_i(t-1)\nu_{T_r}(t-1)^{d-1}}{n^d}. \end{aligned}$$

Based on this probability, we can show

$$\Pr[-\mathcal{E}_{i+1} \mid \mathcal{E}_{i+1} \wedge \mathcal{E}_{T_r}] \leq \frac{\exp(-q_i n)}{\Pr[\mathcal{E}_i \wedge \mathcal{E}_{T_r}]}$$

using an argument similar to the proof of Lemma 5, where we let

$$q_i = c_2 \frac{\alpha_{T_r}^{d-1} \alpha_i}{n^d}. \tag{2}$$

In the range of i satisfying that $\exp(-q_i n) \leq \frac{1}{(\ln n)^2}$,

$$\Pr[-\mathcal{E}_i \mid \mathcal{E}_{T_r}] \leq \frac{i - T_r}{(\ln n)^2 \Pr[\mathcal{E}_{T_r}]},$$

which implies that

$$\Pr[-\mathcal{E}_i] \leq \frac{i - T_r}{(\ln n)^2} + \Pr[-\mathcal{E}_{T_r}].$$

Let i^* be the minimum i such that $\exp(-q_i n) \geq \frac{1}{(\ln n)^2}$. From (2) and the values of α_i and $\alpha_{T_r}^{d-1}$ which were obtained in the second stage, we can imply

$$\begin{aligned} i^* &\leq T_r + \frac{\ln n - \ln \ln(\ln n)^2}{(d-1) \{S_r T_1 \ln \frac{T_1}{ed} - S_r \ln c'_1 - S_r + \ln c'_1\}} + O(1) \\ &= T_r + \frac{\ln n - \ln \ln(\ln n)^2}{(d-1) S_r T_1 (\ln T_1 - \ln d - O(1))} + O(1). \end{aligned}$$

Again as was done in the proof of Lemma 5, we can show that

$$\Pr[\nu_{i+1} \geq e \ln(\ln n)^2] \leq \frac{i^* - T_r + 1}{(\ln n)^2} + \Pr[-\mathcal{E}_{T_r}].$$

Note that

$$\Pr[\nu_{i^*+2} \geq 1 \mid \nu_{i^*+1} \leq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r}] \leq \frac{\Pr[B(n, \frac{e \ln(\ln n)^2}{\alpha_{T_r}} (\frac{\alpha_{T_r}}{n})^d) \geq 1]}{\Pr[\nu_{i^*+1}(n) \leq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r}]}$$

and

$$\Pr\left[B\left(n, \frac{e \ln(\ln n)^2}{\alpha_{T_r}} \left(\frac{\alpha_{T_r}}{n}\right)^d\right) \geq 1\right] = 1 - \left(1 - e \ln(\ln n)^2 \frac{\alpha_{T_r}^{d-1}}{n^d}\right)^n \leq e \ln(\ln n)^2 \left(\frac{\alpha_{T_r}}{n}\right)^{d-1}. \tag{3}$$

Therefore,

$$\Pr[\nu_{i^*+2} \geq 1 \mid \nu_{i^*+1} \leq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r}] \leq \frac{e \ln(\ln n)^2 \left(\frac{\alpha_{T_r}}{n}\right)^d}{\Pr[\nu_{i^*+1}(n) \leq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r}]},$$

which implies that

$$\begin{aligned} \Pr[\nu_{i^*+2} \geq 1] &\leq \Pr[\nu_{i^*+2} \geq 1 \mid \nu_{i^*+1} \leq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r}] \Pr[\nu_{i^*+1} \leq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r}] \\ &\quad + \Pr[-(\nu_{i^*+1} \geq e \ln(\ln n)^2 \wedge \mathcal{E}_{T_r})] \\ &\leq e \ln(\ln n)^2 \left(\frac{\alpha_{T_r}}{n}\right)^{d-1} + \Pr[\nu_{i^*+1} \leq e \ln(\ln n)^2] + \Pr[-\mathcal{E}_{T_r}] \\ &\leq e \ln(\ln n)^2 \left(\frac{\alpha_{T_r}}{n}\right)^{d-1} + \frac{i^* - T_r + 1}{(\ln n)^2} + \Pr[-\mathcal{E}_{T_r}]. \end{aligned}$$

Since the second term is $o(1)$, we can claim that the maximum load is at most $i^* + 1$ with probability at least

$$\begin{aligned} &1 - 4 \exp\left(-\frac{1}{e^d} \left(\frac{d}{T_1}\right)^{T_1} n\right) - (T_r - T_1) \exp\left(-\frac{c_1}{c_1^d} n \left(c_1 e \left(\frac{ed}{T_1}\right)^{T_1}\right)^{S_r}\right) \\ &\quad - \frac{2e \ln \ln n}{c_1^{d-1}} \left(c_1 e \left(\frac{ed}{T_1}\right)^{T_1}\right)^{(d-1)S_r} - o(1) \end{aligned}$$

by Lemmas 4 and 5. This probability is $1 - o(1)$ if $S_r T_1 = o(\ln n / \ln \ln n)$ and $S_r = \omega(\ln \ln \ln n)$. □

4 Lower Bounds

We next show a lower bound of the maximum load by AdC when we have only one threshold T_1 .

Theorem 2. $r = 1$ and $d = o(\ln n / \ln \ln n)$, \dots
 $AdC \dots \Omega\left(\sqrt{\ln n / d \ln \ln n}\right) \dots 1 - o(1)$

Since if $T_1 > \ln n / \ln \ln n$ then we directly obtain a lower bound of $\Omega(\ln n / \ln \ln n)$ by the argument of the one-choice model, we assume that $T_1 \leq \ln n / \ln \ln n$ throughout this proof.

We divide the process of the approximated d -choice balls-and-bins game into two periods: The first is from time 1 to $n/2$ and the second is from $n/2 + 1$ to n . In the first period, we will estimate a lower bound l of the number $\nu_{T_1}^{AdC}(n/2)$ of the “high” bins, namely, the bins of height T_1 or more using approximation by the one-choice model. Next, in the second period, we will estimate the number m of balls that are put into the l “high” bins through the remaining $n/2$ balls, and then estimate a lower bound of the maximum load in the l bins. Note that any balls behave like those in the one-choice model under the condition that they are put into one of the l bins since such l bins are no longer distinguished. Therefore, we can exploit the standard argument to estimate a lower bound T' of the maximum load for l bins and m balls in the one-choice model. We thus obtain a lower bound $T_1 + T'$ of the maximum load achieved by AdC .

First Period: In the first period, we wish to bound $\nu_{T_1}^{AdC}(n/2)$ below by the Poisson approximation of the one-choice model. To compare AdC with $1C$, we consider the following model: For each ball, select d bins uniformly at random and put the ball into the bin which is selected uniformly at random again from the d bins. (Thus this new model is obviously equivalent to $1C(n, n)$.) As before, let us assume that we have the same sequence of dn bins for the new model and AdC . Then, if some bin of height at most $T_1 - 1$ receives a ball in the new model, AdC must put the ball into a bin whose height is also at most $T_1 - 1$. Thus one can conclude that $\nu_{T_1}^{AdC} \geq \nu_{T_1}^{1C(n, n)}$ always holds. Therefore, it holds that

$$\Pr[\nu_{T_1}^{AdC}(n/2) \leq (n/e)p'_{T_1}] \leq \Pr[\nu_{T_1}^{1C}(n/2) \leq (n/e)p'_{T_1}],$$

where, for a random variable X ,

$$p'_{T_1} = \Pr_{X \in (0, \infty)} [X \geq T_1 : X \text{ has the Poisson distribution with mean } 1/2].$$

By Lemmas 2 and 1, we have

$$\Pr[\nu_{T_1}^{1C}(n/2) \leq (n/e)p'_{T_1}] \leq 4 \Pr[B(n, p'_{T_1}) \leq (n/e)p'_{T_1}] \leq 4 \exp(-(1 - 2/e)p'_{T_1} n).$$

Since $p'_{T_1} \geq e^{-1/2} \left(\frac{1}{2T_1}\right)^{T_1}$ by Lemma 3, we have

$$\Pr \left[\nu_{T_1}^{AdC}(n/2) \leq \frac{n}{e^{3/2}} \left(\frac{1}{2T_1}\right)^{T_1} \right] \leq 4 \exp \left(-(1 - 2/e)e^{-1/2} \left(\frac{1}{2T_1}\right)^{T_1} n \right) \stackrel{\text{def}}{=} P_1. \tag{4}$$

Therefore, $\nu_{T_1}^{\text{AdC}}(n/2) > \frac{n}{e^{3/2}} \left(\frac{1}{2T_1}\right)^{T_1}$ with probability at least $1 - P_1$. As mentioned at the beginning of this proof, we denote this lower bound by l , namely,

$$l \stackrel{\text{def}}{=} \frac{n}{e^{3/2}} \left(\frac{1}{2T_1}\right)^{T_1}.$$

Second Period: Let m be the number of balls that are put into the l bins of height T_1 or more through the remaining $n/2$ balls. By Lemma 1,

$$\Pr \left[m \leq \frac{n}{2e} \left(\frac{l}{n}\right)^d \right] \leq \exp \left(\left(\frac{2}{e} - 1\right) \frac{n}{2} \left(\frac{l}{n}\right)^d \right) \stackrel{\text{def}}{=} P_2. \quad (5)$$

By the standard analysis for the one-choice model (e.g., [8]), the lower bound of the maximum load achieved by $1\text{C}(m, l)$ is

$$T' = \Omega \left(\frac{\ln l}{\ln(l/m)} \right) \quad (6)$$

with probability at least $1 - O(1/n)$. We therefore obtain a lower bound by AdC of

$$T_1 + T' = T_1 + \Omega \left(\frac{\ln l}{\ln(l/m)} \right) = \Omega \left(T_1 + \frac{\ln n}{dT_1 \ln T_1} \right) = \Omega \left(\sqrt{\frac{\ln n}{d \ln \ln n}} \right)$$

with probability at least $1 - P_1 - P_2 - O(1/n)$ by (3), (4) and (5). This probability is $1 - o(1)$ if $d = o(\ln n / \ln \ln n)$. \square

5 Concluding Remarks

Recall that the primary motivation of this work was to bridge between the one-choice and two-choice models. In fact, however, our analysis is based on the more general d -choice model, which gives us byproducts such as the interesting property on the merit of increasing d . It should be noted that this also gives us what remains to be done. For example, we assumed that d is constant for our upper bounds. It would be much better if this restriction is removed.

References

1. M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. *Random Structures and Algorithms*, 13(2):159–188, 1998.
2. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 2000.
3. A. D. Barbour, L. Holst, and S. Janson. *Poisson Approximation*. Oxford University Press, 1992.
4. P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: the heavily loaded case. In *Proceedings of the 32nd annual ACM Symposium on Theory of Computing*, pages 745–754, 2000.

5. E. Drinea, A. Frieze, and M. Mitzenmacher. Balls and bins models with feedback. In *Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 308–315, 2002.
6. R. Karp, M. Luby, and F. Meyer auf de Heide. Efficient PRAM simulation on a distributed memory machine. *Algorithmica*, 16(4/5):517–542, 1996.
7. M. Mitzenmacher. Load balancing and density dependent jump markov processes. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 213–222, 1996.
8. M. Mitzenmacher. *The power of two choices in randomized load balancing*. PhD thesis, University of California, Berkeley, 1996.
9. M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.
10. M. Mitzenmacher, B. Prabhakar, and D. Shah. Load balancing with memory. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 799–808, 2002.
11. M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The power of two random choices: a survey of techniques and results. In *Handbook of Randomized Computing*, volume 1, pages 255–312. Kluwer Press, 2001.
12. M. Mitzenmacher and B. Vöcking. The asymptotics of selecting the shortest of two, improved. In *Analytic Methods in Applied Probability: In Memory of Fridrih Karpelevich*, pages 165–176. American Mathematical Society, 2002.
13. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic phase transitions. *Nature*, 400:133–137, 1999.
14. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
15. B. Vöcking. How asymmetry helps load balancing. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 131–140, 1999.
16. B. Vöcking. Symmetric vs. asymmetric multiple-choice algorithms (invited paper). In *Proceedings of the 2nd ARACNE workshop*, pages 7–15, 2001.
17. N. D. Vvendenskaya, R. L. Dobrushin, and F. I. Karpelevich. Queuing systems with selection of the shortest of two queues: An asymptotic approach. *Problems of Information Transmission*, 32(1):15–27, 1996.

Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting

Joseph JaJa¹, Christian W. Mortensen^{2,*}, and Qingmin Shi¹

¹ Institute of Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA

{joseph, qshi}@umiacs.umd.edu

² IT University of Copenhagen, Rued Langgaardsvej 7, 2300 København S, Denmark
cworm@itu.dk

Abstract. We present linear-space sub-logarithmic algorithms for handling the *3-dimensional dominance reporting* and the *2-dimensional dominance counting* problems. Under the RAM model as described in [M. L. Fredman and D. E. Willard. “Surpassing the information theoretic bound with fusion trees”, *Journal of Computer and System Sciences*, 47:424–436, 1993], our algorithms achieve $O(\log n / \log \log n + f)$ query time for the 3-dimensional dominance reporting problem, where f is the output size, and $O(\log n / \log \log n)$ query time for the 2-dimensional dominance counting problem. We extend these results to any constant dimension $d \geq 3$, achieving $O(n(\log n / \log \log n)^{d-3})$ space and $O((\log n / \log \log n)^{d-2} + f)$ query time for the reporting case and $O(n(\log n / \log \log n)^{d-2})$ space and $O((\log n / \log \log n)^{d-1})$ query time for the counting case.

1 Introduction

The d -dimensional dominance reporting (resp. counting) problem for a set S of d -dimensional points is to store S in a data structure such that given a query point q the points in S that dominate q can be reported (resp. counted) quickly. A point $p = (p_1, p_2, \dots, p_d)$ dominates a point $q = (q_1, q_2, \dots, q_d)$ if $p_i \geq q_i$ for all $i = 1, \dots, d$. A number of geometric retrieval problems involving iso-oriented objects can be reduced to these problems (see for example [EO82]). For the rest of the introduction we let n denote the number of points in S , f denote the number of points reported by a dominance reporting query, and $\epsilon > 0$ be an arbitrary small constant. The results of this paper can be summarized by the following two theorems.

Theorem 1. For any constant dimension $d \geq 3$, there exists a data structure for d -dimensional dominance reporting that uses $O(n(\log n / \log \log n)^{d-3})$ space and $O((\log n / \log \log n)^{d-2} + f)$ query time.

* Part of this work was done while the author was visiting the Max-Planck-Institut für Informatik, Saarbrücken, as a Marie Curie doctoral fellow.

Theorem 2. For $d \geq 2$, the query time is $O(n(\log n / \log \log n)^{d-2})$ for reporting and $O((\log n / \log \log n)^{d-1})$ for counting.

Note that a d -dimensional range counting query (in which each coordinate of a point to be reported is bounded from two sides instead of one as in dominance counting) can be handled by combining the results of a constant number (depending on d) of dominance counting queries of the same dimension. Hence the results in Theorem 2 are valid for range counting queries as well.

In this paper, we assume the RAM model as described by Fredman and Willard in [FW93], which allows the construction of q-heaps [FW94] (to be discussed in Section 2). In both Theorems 1 and 2 we assume coordinates of points are integer valued. But for $d \geq 4$ in Theorem 1 and $d \geq 3$ in Theorem 2 our results actually hold for real-valued coordinates.

Our success in proving the above theorems is based on a novel generalization of the concept of dimensions. This definition, given in Section 2, allows a k -dimensional point ($k \leq d$) in the standard sense to be appended with $d - k$ “special” coordinates, each of which can take only $\lceil \log^\epsilon n \rceil$ different values. Our approach is based on the fact that we can prove more general results for k -dimensional reporting (see Lemma 1) and counting queries (see Lemma 4) than what has been done before. That is, in addition to the constraints on the first k coordinates specified by a standard dominance query, our solutions satisfy the additional constraints on the $d - k$ “special” coordinates. These results will in turn lead to efficient extensions to higher dimensions.

1.1 Relation to Prior Work

In [CE87], Chazelle and Edelsbrunner proposed two linear-space algorithms for the 3-dimensional dominance reporting problem. The first achieves $O(\log n + f \log n)$ query time and the second achieves $O(\log^2 n + f)$ query time. These two algorithms were later improved by Makris and Tsakalidis [MT98] to yield $O((\log \log U)^2 \log \log \log U + f \log \log U)$ query time for a special case where coordinates of the points are integers from a bounded universe $[0, \dots, U]$ and $O(\log n + f)$ query time for the general case. The previous best linear-space algorithm for the 2-dimensional dominance counting problem is due to Chazelle [Cha88] and achieves $O(\log n)$ query time. An external version of Chazelle’s scheme was proposed by Govindarajan, Arge, and Agarwal [GAA03], which achieves linear space and $O(\log_B n)$ I/Os (B being the disk block size) for handling range counting queries in \mathcal{R}^2 . They also extended their algorithm to handle higher dimensional range counting queries, introducing a factor of $O(\log_B n)$ to both the space and the query complexity for each additional dimension.

In [SJ03a], Shi and JaJa achieved $O(\log n / \log \log n + f)$ query time for the reporting case and $O(\log n / \log \log n)$ query time for the counting case, but at the expense of increasing the space cost in both cases by a factor of $\log^\epsilon n$. Like our solution, these solutions require coordinates of points to be integers.

It follows that, compared with previous solutions that require linear space and, for the reporting case, constant time per reported point, our results improve

the query time by a factor of $\log \log n$. Further, the standard techniques for extending the structures into higher dimensions either require a $\log n$ factor on both the query time and the space usage for each dimension [Ben80] or a $\log^{1+\epsilon} n$ factor on the space usage and a $\log n / \log \log n$ factor on the query time for each dimension [ABR00] (while still only using constant time for each reported point). In this paper, we improve the cost of extensions to higher dimensions to a factor of $O(\log n / \log \log n)$ per dimension for both query time and space usage.

1.2 Paper Outline

We start with preliminaries in Section 2. In Section 3 we give a solution to our generalized 3-dimensional dominance reporting problem, thus proving Theorem 1 for $d = 3$. In Section 4 we give a solution to our generalized 2-dimensional dominance counting problem and prove Theorem 2 for $d = 2$. Finally, we extend our results to higher dimensions in Section 5.

2 Preliminaries

If an array A is indexed by a set M we will use the notation $A[m]$ for the element of A at index $m \in M$. For integers i and j we let $[i..j]$ denote the set of integers k for which $i \leq k \leq j$. If M is a set and $k \geq 0$ is an integer we let M^k denote the set $M \times \dots \times M$ where M is repeated k times. When stating theorems, we define $i/0 = \infty$ when $i > 0$.

For the rest of this paper, we assume n is the number of points for the dominance reporting or counting structure we are ultimately designing. While developing the ultimate structures, we will construct structures with fewer than n points. We will assume we can use time $O(n)$ to precompute a constant number of tables with size $O(n)$ which only depend on the word size and on n .

We say a point $p = (p_1, \dots, p_d)$ has dimension (d', d, ϵ) if $p_{d+1}, \dots, p_d \in [1..[\log^\epsilon n]]$. Here we assume $1 \leq d' \leq d$ and $0 < \epsilon < 1$ are all constants. We refer to p_i , $1 \leq i \leq d$, as the i -coordinate of p . We say a set S has dimension (d', d, ϵ) if all the elements of S are (d', d, ϵ) -dimensional points. When creating a data structure containing a set S with dimension (d', d, ϵ) , we assume without loss of generality that no two different points in S share any of their first d' coordinates. That is, if $p = (p_1, \dots, p_d) \in S$ and $r = (r_1, \dots, r_d) \in S$ and $p \neq r$ then $p_1 \neq r_1, \dots, p_{d'} \neq r_{d'}$. We define the i -coordinate of p as the point $p = (p_1, \dots, p_d) \in S$ such that $p_i \geq s$ and p_i is minimized. We define the i -coordinate of p as the number of points $(p_1, \dots, p_d) \in S$ for which $p_i \leq s$, and the i -rank of p in S as the i -rank of its i -coordinate. We say the i -coordinate is in S if for all points $p = (p_1, \dots, p_d) \in S$ the i -rank of p_i is equal to p_i .

We will use the q-heap data structure of Fredman and Willard [FW94]. A q-heap allows a set S of words where S has cardinality $O(\log^{1/4} n)$ to be stored such that elements can be inserted in and deleted from S in constant time and such that given a query integer q the largest element of S smaller than q can be identified in constant time. The q-heap requires a precomputed table with size $O(n)$ computable in $O(n)$ time. The size of this table only depends on the word size and on n .

3 Three-Dimensional Dominance Reporting

The purpose of this section is to design a data structure to solve the dominance reporting problem for a set with dimension $(3, d, \epsilon)$ for $d \geq 3$, in particular proving Theorem 1 for $d = 3$. We give this structure in Section 3.2. First, we give, in Section 3.1, a solution to a variant of the dominance reporting problem for a set with dimension $(2, d, \epsilon)$, where $d \geq 2$.

3.1 $(2, d, \epsilon)$ -Dimensional 3-Sided Reporting

We define the $(2, d, \epsilon)$ -dimensional 3-sided reporting problem for a set S with dimension $(2, d, \epsilon)$ as a generalization of the dominance reporting problem as follows. A query must have the form $((q_1, q'_1), q_2, \dots, q_d)$ where $q_1 \leq q'_1$. The answer to such a query is the set of points $p = (p_1, \dots, p_d) \in S$ for which $q_1 \leq p_1 \leq q'_1$ and $p_2 \geq q_2, \dots, p_d \geq q_d$. We have:

Lemma 1. *Let $d \geq 2$ and $0 < \epsilon < 1/(d-2)$. Let S be a set of $m \leq n$ points in \mathbb{R}^d with dimension $(2, d, \epsilon)$. Let $Q = ((q_1, q'_1), q_2, \dots, q_d)$ be a query. Then the number of points in S that are reported by Q is $O(m)$. The time to report Q is $O(1 + f)$, where f is the number of points reported by Q .*

For $d = 2$ this problem is well studied (see the survey by Alstrup et al. [AGKR02]). We show the lemma for any $d \geq 2$ by using an extension of a technique mentioned in [AGKR02]. Actually, we show Lemma 1 as a corollary to Lemma 2:

Lemma 2. *Let $m \leq n$ and $0 < \epsilon < 1/(d-2)$. Let S be a set of m points in \mathbb{R}^d with dimension $(2, d, \epsilon)$. Let $Q = ((q_1, q'_1), q_3, \dots, q_d)$ be a query. Let $p = (p_1, \dots, p_d) \in S$ be the point we get. If $p_2 < q_2$ we stop. Otherwise, we report p and recursively ask the two queries $((q_1, p_1 - 1), q_3, \dots, q_d)$ and $((p_1 + 1, q'_1), q_3, \dots, q_d)$ in Lemma 2.*

We obtain Lemma 1 from Lemma 2 as follows. Suppose we are given a query $((q_1, q'_1), q_2, \dots, q_d)$ in Lemma 1. We then ask the query $((q_1, q'_1), q_3, \dots, q_d)$ in Lemma 2. If we do not get a point we stop, else let $p = (p_1, \dots, p_d) \in S$ be the point we get. If $p_2 < q_2$ we stop. Otherwise, we report p and recursively ask the two queries $((q_1, p_1 - 1), q_3, \dots, q_d)$ and $((p_1 + 1, q'_1), q_3, \dots, q_d)$ in Lemma 2.

The rest of this section is devoted to the proof of Lemma 2. We assume m is a power of two and define $c = \lceil \log^\epsilon n \rceil$. We sort the points in increasing order by their 1-coordinates and group them into blocks each with $c^{d-2} \log m$ elements. We define for each block b a range $[b.l..b.r]$, where $b.l$ (resp. $b.r$) is the smallest (resp. largest) 1-coordinate of a point in b . We create a binary tree T built on the blocks b sorted in increasing order by $b.l$. We associate with each leaf v of T two arrays $v.left$ and $v.right$, both indexed by $[0.. \log m] \times [1..c]^{d-2}$. Let u be the ancestor of v whose height is h (the height of a leaf is 0). Then $v.left[h, q_3, \dots, q_d]$ (resp. $v.right[h, q_3, \dots, q_d]$) stores the point $p = (p_1, \dots, p_d)$ such that (i) p belongs to a block corresponding to a leaf between v and the leftmost (resp. rightmost) leaf node of the subtree rooted at u ; (ii) $p_3 \geq q_3, \dots, p_d \geq q_d$; and (iii) p_2 is

maximized, provided that such a point p exists. We note that, since each of the $O(m/(c^{d-2} \log m))$ leaves of T contains two arrays both with $O(c^{d-2} \log m)$ elements, the total space used so far is $O(m)$.

The data structure we just described has enabled us to answer a query $((q_1, q'_1), q_3, \dots, q_d)$ where $q_1 = b.l$ and $q'_1 = b'.r$ for two blocks $b \neq b'$. We first locate, in constant time, the nearest common ancestor u of the two leaves corresponding to b and b' . This can be done in constant time using table lookup. Let h be the height of u . Then the point that satisfies the query (if any) must be in either $b.right[h - 1, q_3, \dots, q_d]$ or $b'.left[h - 1, q_3, \dots, q_d]$.

In order to be able to answer general queries where the range $[q_1, q'_1]$ may partially intersect the 1-ranges of some blocks, we create an extra structure for each block b as follows. We build a constant-height tree $b.T$ with degree $\Theta(\log^\delta n)$, for a constant $\delta > 0$ to be determined later, over the points of b sorted in increasing order by their 1-coordinates. At each internal node $v \in b.T$ we keep for each pair (v_1, v_2) of its children, where v_1 is to the left of v_2 , an array $v.R_{(v_1, v_2)}$ indexed by $[1..c]^{d-2}$. The entry (q_3, \dots, q_d) of this array identifies the point $p = (p_1, \dots, p_d)$ in b such that (i) p is in a leaf below a child of v between v_1 and v_2 ; (ii) $p_3 \geq q_3, \dots, p_d \geq q_d$; and (iii) p_2 is maximized, provided that such a point exists. Since within b a point can be uniquely identified using $O(\log \log n)$ bits, the total bit-cost of each internal node of $v.T$ is $O(c^{d-2} \log \log n \log^{2\delta} n)$. It follows that an internal node fits into a single word (and thus the total space usage will be linear), if $2\delta + (d - 2)\epsilon < 1$, which can be satisfied by choosing $\delta = (1 - (d - 2)\epsilon)/3$. Our assumption $\epsilon < 1/(d - 2)$ ensures that δ will be positive.

We now describe how to answer a query of the form $((q_1, q'_1), q_3, \dots, q_d)$ where $q_1, q'_1 \in [b.l..b.r]$ for some block b . Let u be the nearest common ancestor of the leaves in $b.T$ corresponding to q_1 and q'_1 , and let Π_1 and Π_2 be respectively the paths from u to these two leaves. The answer to the query can be chosen from a constant number of “candidate points”, each picked at a node on Π_1 or Π_2 from one of its associated arrays. For example, without loss of generality, consider the node u . Suppose its i th and j th children are respectively on paths Π_1 and Π_2 , and assume $i < j - 1$. Then the candidate point contributed by u can be found in constant time at $u.R_{(v_1, v_2)}[q_3, \dots, q_d]$, where v_1 and v_2 are respectively the $(i + 1)$ th and $(j - 1)$ th children of u .

Finally suppose we are given a query $((q_1, q'_1), q_3, \dots, q_d)$ where $q_1 \in [b.l..b.r]$ and $q'_1 \in [b'.l..b'.r]$ for two different blocks b and b' , which can be easily identified in constant time. The output of the query is then one of the three points returned by the queries $((q_1, b.r), q_2, \dots, q_d)$, $((b'.l, q'_1), q_2, \dots, q_d)$, and $(b.r + 1, b'.l - 1, q_2, \dots, q_d)$, the handling of which has already been described.

3.2 (3, d, ε)-Dimensional Dominance Reporting

In this section we show the following lemma:

Lemma 3. . . . $d \geq 3$. . . $0 < \epsilon < \min(1/4, 1/(d - 2))$
 S . . . $(3, d, \epsilon)$ $m \leq n$
 S $O(m)$
 f $O(\log m / \log \log n + f)$

We say a point $p = (p_1, \dots, p_d)$ 2-dominates a point $q = (q_1, \dots, q_d)$ if $p_i \geq q_i$, for $1 \leq i \leq 2$. We say a subset M of a point set P is 2-maximal if $p \in M$, $q \in P$ and $p \neq q$ implies that p does not 2-dominate q .

We build a search tree T with degree $c = \lceil \log^\epsilon n \rceil$ and height $O(\log m / \log \log n)$ over the points from S sorted in increasing order by their 3-coordinates. At each internal node $v \in T$ we store the keys to guide the search in T in a q-heap. For each internal or leaf node $v \in T$ we define $M(v)$ to be the largest 2-maximal set of the points stored in the leaves of the subtree rooted at v , excluding those in $M(v')$ for any ancestor v' of v . We keep each point $(p_1, \dots, p_d) \in M(v)$ as the $(2, d-1, \epsilon)$ -dimensional point $(p_1, p_3, p_4, \dots, p_d)$ in the 3-sided reporting structure $D(v)$ of Lemma 1. For the moment we ignore the requirement in Lemma 1 that the 1-coordinate must be in rank space. At each internal node v , we also store another data structure $G(v)$ containing the points in $M(v')$ for all the children v' of v . A point (p_1, \dots, p_d) from the i -child of v (counted from the left) is stored in $G(v)$ as the $(2, d, \epsilon)$ -dimensional point $(p_1, p_2, i, p_4, \dots, p_d)$. $G(v)$ is also a 3-sided reporting structure of Lemma 1, again ignoring the rank space requirement.

Now suppose we are given a query $q = (q_1, \dots, q_d)$. We first identify the path Π in T from the root to the leaf corresponding to the 3-successor of q_3 . We can find Π in time $O(\log m / \log \log n)$ by using the q-heaps associated with the nodes of T . We then visit the root of T as described in the following. Actually, we will describe how to visit an arbitrary node $v \in T$. We first report the points from $M(v)$ which dominate q by performing a query $q' = ((q_1, q'_1), q_3, q_4, \dots, q_d)$ in $D(v)$, where q'_1 is the 1-coordinate of the 2-successor of q_2 in $M(v)$ (we will explain later how to find this 2-successor in constant time). The points returned by this query are exactly the points in $M(v)$ that dominate q . This is due to the fact that for any two points $r = (r_1, r_2, \dots, r_d)$ and $s = (s_1, s_2, \dots, s_d)$ in $M(v)$, $r_1 > s_1$ if and only if $r_2 < s_2$ (see [MT98] for more details). Next, suppose the k th child of v is on Π (we set $k = 0$ if v is not on Π). We then perform the query $q'' = ((q_1, \infty), q_2, k+1, q_3, \dots, q_d)$ in $G(v)$. If the answer to q'' contains a point $(p_1, p_2, i, p_4, \dots, p_d)$, we recursively visit the i th child of v . If v has a child on Π we also recursively visit that child (such child exists if and only if v is an internal node on Π).

We now address the issue that the 1-coordinate of the points in $M(v)$ and $G(v)$ for a node v in T has to be in rank space in order for Lemma 1 to be applicable, and that the 2-successor of q_2 in $M(v)$ has to be identified in constant time. The first issue is resolved by replacing the 1-coordinate of each point in $M(v)$ (resp. $G(v)$) with its 1-rank with respect to $M(v)$ (resp. $G(v)$). Accordingly, before the query q' (resp. q'') is applied to $M(v)$ (resp. $G(v)$) we replace q_1 with its 1-rank in $M(v)$ (resp. $G(v)$) (and, in the case of query q' , replace q'_1 with the 1-rank of the 2-successor of q_2). Computing the 1-rank and the 2-successor of an integer in $M(v)$ and $G(v)$ for each node visited is exactly the problem defined in [CG86], which can be handled in $O(1)$ time if v is not the root of T and in $O(\log m / \log \log n)$ time if v is the root, using the fast fractional cascading technique of [SJ03b], which requires $O(m)$ space. (Notice

that the standard fractional cascading technique described in [CG86] will not work here because the degree of T is not a constant.)

Note that a point in S is stored at most twice, once in $D(v)$ for a node v and once in $G(u)$, where u is the parent of v . The data structures of types D and G are all linear-space data structures. Therefore the overall space usage is $O(m)$. Further, it is easy to see that the number of nodes of T visited is $O(\log m / \log \log n + f)$ and that searching $D(v)$ and $G(v)$ at each such node v takes $O(1)$ time per reported point, hence the claimed querying complexity.

4 Two-Dimensional Dominance Counting

The goal of this section is to design a data structure to solve the dominance counting problem for a set with dimension $(2, d, \epsilon)$ for $d \geq 2$, thus proving Theorem 2 for $d = 2$. We give this structure in Section 4.2. But first, we give, in Section 4.1, a solution with sub-linear space usage for a set with dimension $(1, d, \epsilon)$, for $d \geq 1$, where the 1-coordinate is in rank space.

4.1 $(1, d, \epsilon)$ -Dimensional Dominance Counting

This section is devoted to the proof of the following lemma:

Lemma 4. *Let $d \geq 1$ and $0 < \epsilon < 1/(d - 1)$. For a set S of dimension $(1, d, \epsilon)$ with $m \leq n$ points, there is a data structure for S with space $O(m \log \log n)$ and query time $O(\log n)$.*

We assume m is a power of two and define $c = \lceil \log^\epsilon n \rceil$. We sort the points in increasing order by their 1-coordinates and group them into blocks each with $c^{d-1} \log m$ elements. Further, we partition each block into subblocks each with c^{d-1} elements. We label each block (resp. subblock) with the largest 1-coordinate of a point in that block (resp. subblock). For each block or subblock b we keep an array $b.\text{count}$ indexed by $[1..c]^{d-1}$. For each block b we set $b.\text{count}[q_2, \dots, q_d]$ to be the number of points in S dominating $(i + 1, q_2, \dots, q_d)$ where i is the label of b . For each subblock b' of a block b we set $b'.\text{count}[q_2, \dots, q_d]$ to be the number of points in b dominating $(i + 1, q_2, \dots, q_d)$ where i is the label of b' . Finally, we encode the points of a subblock b' in $O(c^{d-1} \log c) = o(\log n)$ bits which we keep in a single word. This is possible since each of the c^{d-1} points in b' can be encoded in $O(\log c)$ bits.

Suppose we are given a query $q = (q_1, \dots, q_d)$. We first identify, in constant time, the block (resp. subblock) b (resp. b') with the smallest label greater than or equal to q_1 . We now describe how to find the number e of points in b' that dominate q . Notice that a query q with respect to a subblock can be described in $O(\log \log n)$ bits. To compute e in constant time, all we need is to append q to the description of b' and use the result to look up a global table, which requires $O(n)$ words since $O(c^{d-1} \log c \log \log n) = o(\log n)$. The answer to the query q is then $e + b.\text{count}(q_2, \dots, q_d) + b'.\text{count}(q_2, \dots, q_d)$.

We now analyze the space usage of the structure. Each array of the $m/(c^{d-1} \log m)$ blocks contains c^{d-1} elements each of $\log m$ bits. It follows that the space used by these arrays is $O(m)$ bits. Each array of the m/c^{d-1} subblocks also contains c^{d-1} elements but each element is at most $c^{d-1} \log m$. It follows that each element can be represented by $O(\log \log n)$ bits so the total space used by the arrays associated with the subblocks is $O(m \log \log n)$ bits. Finally, each point of a subblock can be represented by $O(\log \log n)$ bits and it follows that the total space usage becomes $O(m \log \log n)$ as claimed.

4.2 (2, d, ε)-Dimensional Dominance Counting

In this section we show:

Lemma 5. *Let $d \geq 2$, $0 < \epsilon < \min(1/4, 1/(d - 1))$, $m \leq n$ and S be a set of m points in \mathbb{R}^d . Then there exists a structure D of size $O(m)$ that can answer $(2, d, \epsilon)$ -dimensional dominance counting queries in $O(\log m / \log \log n)$ time.*

We will prove the lemma under the assumption that the first coordinate is in rank space. Since the targeted query time is $O(\log m / \log \log n)$ time, we can easily remove this assumption by transforming the 1-coordinates of the points in S as well as the 1-coordinate of the query point into rank space by creating a search tree with degree $(\log^\delta n)$ on the 1-coordinates of the points, where $\delta < 1/4$, and storing the keys at each node of this tree in a q-heap.

We create a search tree T with degree $c = \lceil \log^\epsilon n \rceil$ and height $O(\log m / \log \log n)$ over the points from S sorted in increasing order by their 2-coordinates. At each internal node $v \in T$ we store the keys to guide the search in v in a q-heap. We define $G(v)$ to be the set of points stored in the subtree rooted at v . Let $p = (p_1, \dots, p_d) \in G(v)$ be a point stored at a leaf descendant of the j th child of v , and let i be the 1-rank of p in $G(v)$. Then we store p as (i, j, p_3, \dots, p_d) in a structure $v.D$ of Lemma 4 with dimension $(1, d, \epsilon)$.

Now assume that we are given a query $q = (q_1, \dots, q_d)$. We first identify the path Π in T from the root to the leaf storing the 2-successor of q_2 . We can find Π in time $O(\log m / \log \log n)$ by using the q-heaps stored at the nodes of T . The answer to q is the sum of the answers to $\Theta(\log m / \log \log n)$ $(1, d, \epsilon)$ -dimensional dominance counting queries, each applied to the structure $v.D$ for a node v on Π . For the root w , suppose the j th child of w is also on Π . Then the query applied to $w.D$ is $(q_1, j + 1, q_3, \dots, q_d)$. For the leaf that is on Π , we check the point stored there directly. Now consider a non-root internal node v on Π . Let u be its parent. (Note that u is also on Π .) Suppose the j th child of v is also on Π . Note that we already know the 1-rank $r_1(u)$ of q_1 in $G(u)$ ($r_1(w) = q_1$). The query applied to $v.D$ is $(r_1(v), j + 1, q_3, \dots, q_d)$, where $r_1(v)$ is the 1-rank of q_1 in $G(v)$. The value of $r_1(v)$ can be computed by performing a constant number of $(1, d, \epsilon)$ -dimensional dominance queries in $u.D$. In fact, let $G_{>}(v)$ denote the union of $G(v')$ for all the right siblings v' of v , and let $G_{\geq}(v) = G(v) \cup G_{>}(v)$. Then $r_1(v)$ is the difference between the 1-rank of q_1 in $G_{\geq}(v)$ and the 1-rank of q_1 in $G_{>}(v)$. The 1-rank of q_1 in $G_{\geq}(v)$ is $|G_{\geq}(v)| - k_{\geq}(v) + 1$, where $k_{\geq}(v)$

is the number of points in $G_{\geq}(v)$ whose 1-coordinate is greater than or equal to q_1 . $|G_{\geq}(v)|$ can be computed by performing the query $(0, j, q_3, \dots, q_d)$ in $u.D$ and $k_{\geq}(v)$ can be computed by performing the query $(r_1(u), j, q_3, \dots, q_d)$ in $u.D$. The 1-rank of q_1 in $G_{>}(v)$ can be computed similarly.

Since we only use constant time at each node of H , the overall query time is $O(\log m / \log \log n)$. Furthermore, we store each of the m points of S in $O(\log m / \log \log n)$ structures, each of which uses $O(\log \log n)$ bits. Therefore the total space usage becomes $O(m)$ as claimed.

5 Higher Dimensional Dominance Reporting and Counting

In this section we give a general construction in Lemma 6 which can be used to obtain a structure for a $(d, d, 0)$ -dimensional point set from a structure for a (d', d, ϵ) -dimensional point set, where $d' < d$. We then use this construction to prove Theorem 1 and 2 from Lemma 3 and 5 respectively. A similar construction was given in [Mor03] for orthogonal range reporting for a dynamic set of points.

Let $(G, +)$ be a semi-group and let S be a set with dimension (d', d, ϵ) . Assume each point $p \in S$ has an associated semigroup element $g(p) \in G$. The (d', d, ϵ) -dominance reporting problem is defined as follows. Given a (d', d, ϵ) -dimensional query point q , find the semigroup sum $\sum_{p \in S: p \text{ dominates } q} g(p)$. We will show:

Lemma 6. Let $2 \leq d' \leq d$, $0 < \epsilon < 1/4$, and let D be a structure for a (d', d, ϵ) -dimensional point set of size m . Then there exists a structure D' for a $(d, d, 0)$ -dimensional point set of size m' such that $m' \leq n$ and D' can answer dominance reporting queries in $O(\log m / \log \log n)$ time. Furthermore, D' can answer dominance counting queries in $O(\log m / \log \log n)$ time. The total space usage of D' is $O(n)$.

The dominance reporting problem can be seen as a special case of the dominance semigroup problem if we define the elements in G as point sets, $g(p) = \{p\}$, and select $+$ to be the union operator on sets. Theorem 1 then follows by applying Lemma 6 $d-3$ times to Lemma 3. Similarly, the dominance counting problem can be seen as a special case of the dominance semigroup problem, where the elements in G are non-negative integers, $g(p) = 1$, and $+$ is the integer addition. Theorem 2 then follows by applying Lemma 6 $d-2$ times to Lemma 5.

We now prove Lemma 6. Let S be a set of m (d', d, ϵ) -dimensional points. We build a tree T with degree $c = \lceil \log^\epsilon m \rceil$ over the points in S sorted by their

d' -coordinates. At each internal node $v \in T$ we keep a dominance semigroup structure $v.D'$ with dimension $(d' - 1, d, \epsilon)$ containing the points stored in the subtree rooted at v . A point $p = (p_1, \dots, p_d)$ from the i th child of v is stored in $v.D'$ as $p' = (p_1, \dots, p_{d'-1}, i, p_{d'+1}, \dots, p_d)$ with $g(p') = g(p)$. Suppose now we are given a query $q = (q_1, \dots, q_d)$ in D with dimension (d', d, ϵ) . We first identify the path Π in T from the root to the leaf corresponding to the d' -successor of $q_{d'}$, which can be found in time $O(\log n / \log \log n)$ using the q-heaps in the nodes of T . For each internal node $v \in \Pi$, assume that the j_v th child of v is also on Π . The answer to the query in D is then the semigroup sum of the queries $(q_1, \dots, q_{d'-1}, j_v + 1, q_{d'+1}, \dots, q_d)$ in $v.D'$ for every $v \in \Pi$. This finishes the proof of Lemma 6.

References

- [ABR00] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 198–207, Redondo Beach, CA, 2000.
- [AGKR02] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: A survey and a new distributed algorithm. In *Proceedings of the 14th ACM Symp. on Parallel Algorithms and Architecture (SPAA)*, pages 258–264, August 2002.
- [Ben80] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, April 1980.
- [CE87] Bernard Chazelle and H. Edelsbrunner. Linear space data structures for two types of range search. *Discrete Comput. Geom.*, 3:113–126, 1987.
- [CG86] Bernard Chazelle and Leonidas J. Guibas. Fractional Cascading: I. A data structure technique. *Algorithmica*, 1(2):133–162, 1986.
- [Cha88] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–463, June 1988.
- [EO82] H. Edelsbrunner and M.H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14:124–127, 1982.
- [FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.
- [FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48:533–551, 1994.
- [GAA03] Sathish Govindarajan, Pankaj K. Agarwal, and Lars Arge. CRB-Tree: an efficient indexing scheme for range-aggregate queries. In *Proceedings of the 9th International Conference on Database Theory*, Siena, Italy, 2003.
- [Mor03] Christian Worm Mortensen. Fully-dynamic orthogonal range reporting on RAM (Preliminary version). Technical Report TR-2003-22, The IT University of Copenhagen, 2003.
- [MT98] C. Makris and A. K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.

- [SJ03a] Qingmin Shi and Joseph JaJa. Fast algorithms for 3-d dominance reporting and counting. Technical Report CS-TR-4437, Institute of Advanced Computer Studies (UMIACS), University of Maryland, 2003.
- [SJ03b] Qingmin Shi and Joseph JaJa. Fast fractional cascading and its applications. Technical Report CS-TR-4502, Institute of Advanced Computer Studies (UMIACS), University of Maryland, 2003.

Local Gapped Subforest Alignment and Its Application in Finding RNA Structural Motifs

Jesper Jansson, Ngo Trung Hieu, and Wing-Kin Sung

School of Computing, National University of Singapore, 3 Science Drive 2,
Singapore 117543

{jansson, ngotrung, ksung}@comp.nus.edu.sg

Abstract. We consider the problem of computing an optimal local alignment of two labeled ordered forests F_1 and F_2 where n_i and d_i , for $i \in \{1, 2\}$, denote the number of nodes in F_i and the degree of F_i , respectively; and its applications in finding RNA structural motifs. A previous result is the local closed subforest alignment problem, which can be solved in $O(n_1 n_2 d_1 d_2 (d_1 + d_2))$ time and $O(n_1 n_2 d_1 d_2)$ space. This paper generalizes the concept of a closed subforest to a *gapped subforest* and then presents an algorithm for computing the optimal local *gapped subforest alignment* of F_1 and F_2 in $O(n_1 n_2 d_1 d_2 (d_1 + d_2))$ time and $O(n_1 n_2 d_1 d_2)$ space. We show that our technique can improve the computation of the optimal local closed subforest alignment in $O(n_1 n_2 (d_1 + d_2)^2)$ time and $O(n_1 n_2 (d_1 + d_2))$ space. Furthermore, we prove that a special case of our local gapped subforest alignment problem is equivalent to a problem known in the literature as the local sequence-structure alignment problem (*lssa*). The previously best algorithm for *lssa* uses $O(n_1^2 n_2^2 (n_1 + n_2))$ time and $O(n_1 n_2)$ space; here, we show how to modify our main algorithm to obtain an algorithm for *lssa* running in $O(n_1 n_2 (d_1 + d_2)^2)$ time and $O(n_1 n_2 (d_1 + d_2))$ space.

1 Introduction

Many areas of computer science use labeled ordered trees to represent hierarchically structured information. It is often useful to measure the similarity between two or more such trees or to find parts of the trees that are similar. In computational molecular biology, labeled ordered trees are used to represent RNA molecules' secondary structures [14]. By measuring and comparing the similarity of secondary structure trees, researchers who investigate structural or evolutionary relationships between RNA molecules may obtain additional clues [4]. Furthermore, comparing RNAs lead to automated methods for discovering frequently recurring patterns in their secondary structures (also known as *motifs*) which are helpful when investigating the various functions in the cell of different types of RNA [8] or when predicting the secondary structure of a newly found RNA molecule.

Two ways to measure the similarity between two labeled ordered trees are by using the *edit distance* [15] or *tree edit distance* [11]. The problem of computing the optimal alignment of two trees can be viewed as a special

case of the tree edit distance problem [11]; indeed, the fastest known algorithms for optimal alignment between two trees have lower time complexities than the fastest known algorithms for the tree edit distance, both for unordered trees whose degrees are bounded by a constant [11, 19] and for ordered trees whose degrees are much smaller than their depths [11, 20].

Alignments between trees as defined in [11] consider similarities on a node level only, in the sense that a node in the input trees must be paired off with either a node in the other tree or a space. [8] and [16] extended the concept of a global alignment of trees to a local alignment of trees by introducing problems in which the objective is to find two substructures of the input having the highest possible similarity, where the similarity between two substructures is defined using the maximum score of a global alignment between them.

In this paper, we improve the time and space complexities of the main algorithm presented in [8]. Moreover, we further extend the set of mathematical definitions and notations for local similarity in labeled forests by generalizing the concept of a closed subforest used in [8] to what we call a *local similarity forest*. Based on this new concept, we define a computational problem called the *local similarity forest problem* (*lgsf*) that can express even more general patterns of local similarities in two labeled ordered forests than the problem considered in [8], and give an efficient algorithm for solving it. Finally, we prove that a special case of *lgsf* referred to as *lgsf_β* is in fact equivalent to the *local similarity forest problem* (*lssa*) presented in [2], implying that a slightly modified version of our algorithm for *lgsf* can be applied to solve *lssa* much faster than the algorithm given in [2].

1.1 Problem Definitions

We first introduce some terminologies and notations used in this paper.

Let Σ be a set of symbols, the Σ -labeled forest. A rooted ordered forest whose nodes are labeled by symbols in Σ is called a Σ -labeled forest (in short, forest). For any two nodes u and v in F , u and v are called siblings if and only if they have the same parents or both of them are roots of some trees in F . For any node u in F , let $e(u)$ be the rightmost sibling of u ; let $l(u)$ and $r(u)$ be the sibling immediately to the left and to the right of u , respectively. u_L and u_R denote the leftmost and the rightmost children of u respectively. Given two siblings u and v , define $u..v$ as a sibling interval, i.e., the set of siblings which are to the right of u and to the left of v . If u and v are not siblings or if u is a right sibling of v , $u..v$ is an empty interval. Let $S(F)$ be the set of all sibling intervals of F . Note that the number of sibling intervals in $S(F)$ is $O(|F|deg(F))$ since, for each node u , there are at most $deg(F)$ sibling intervals $u..v$. Let $F[u..v]$ be the forest consisting of the subtrees rooted at the nodes in $u..v$.

Definition 1 (Closed Subforest). Let F and F' be Σ -labeled forests. A subforest F' of F is called a closed subforest if $F' = F[u..v]$ for some sibling interval $u..v$ in $S(F)$.

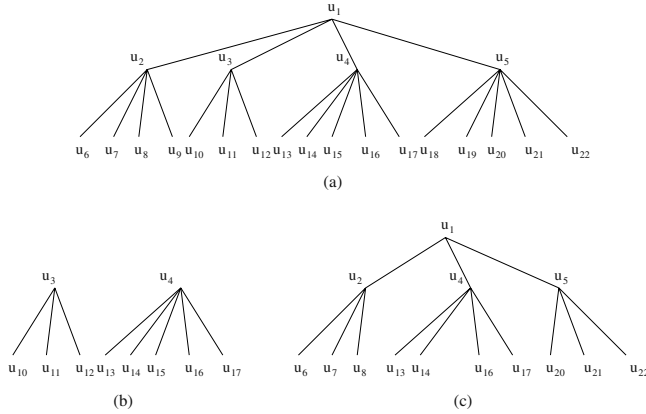


Fig. 1. (a) is an example of tree/forest F ; (b) shows the closed subforest $F[u_{3..u_4}]$; (c) shows a gapped subforest at $F[u_{1..u_1}]$. This gapped subforest is formed by excluding the closed subforests $F[u_{3..u_3}]$, $F[u_{9..u_9}]$, $F[u_{15..u_{15}}]$, and $F[u_{18..u_{19}}]$. The forest in (b) is an α -gapped subforest at $F[u_{3..u_5}]$ since it only excludes $F[u_{5..u_5}]$. Also, (b) and (c) are β -gapped subforests at $F[u_{3..u_4}]$ and $F[u_{1..u_1}]$, respectively

Definition 2 (Gapped Subforest).

Let F and F' be Σ -labeled forests. Let $u..v$ be a node sequence in F . Let $x_i..y_i$ be node sequences in F' for $i = 1, 2, \dots, k$. Let p_i be a node in F' for $i = 1, 2, \dots, k$. Let $p_i \neq p_j$ for $i, j = 1, 2, \dots, k$. Let F' be a forest in F such that $F[x_i..y_i] \subseteq F[u..v]$ for $i = 1, 2, \dots, k$. Let $F[u..v]$ be a forest in F . Let $gsf(F[u..v])$ be the gapped subforest of F at $F[u..v]$ formed by excluding the closed subforests $F[x_i..y_i]$ for $i = 1, 2, \dots, k$. Let $gsf_*(F[u..v])$ be the gapped subforest of F at $F[u..v]$ formed by excluding the closed subforests $F[x_i..y_i]$ for $i = 1, 2, \dots, k$. Let $gsf_\alpha(F[u..v])$ be the α -gapped subforest of F at $F[u..v]$ formed by excluding the closed subforest $F[x..y]$ for $x = u$. Let $gsf_\beta(F[u..v])$ be the β -gapped subforest of F at $F[u..v]$ formed by excluding the closed subforest $F[x..y]$ for $x..y \subseteq u..v$. Let $gsf_\beta(F[u..v])$ be the β -gapped subforest of F at $F[u..v]$ formed by excluding the closed subforests $F[x..y]$ for $x..y \subseteq u..v$.

Lemma 1. $gsf(F[u..u']) = gsfa(F[u..u']) \cup (\cup_{u'' \in u..u'} gsfb[u''..u'])$

Given two Σ -labeled forests F and G , one way to measure their similarity is to compute their optimal global alignment score. The definition of global alignment follows Jiang et al. [11]. Basically, a global alignment \mathcal{A} of F and G is obtained by first inserting nodes labeled with spaces '-' into F and G such that the two resulting ordered forests F' and G' have the same structure, and then overlaying them. Here is the formal definition of the global alignment.

Definition 3 (Global Forest Alignment).

Let F, G be Σ -labeled forests. Let \mathcal{A} be a global alignment of F and G . Let $F' = \pi(A_1)$ and $G' = \pi(A_2)$. Let A_1 and A_2 be the ordered forests obtained by inserting spaces '-' into F and G respectively.

$$\mathcal{A} = \{A_i \mid \pi(A_i) \text{ is a global alignment of } F \text{ and } G, i = 1, 2\}$$

For every pair of labels $(u, v) \in (\Sigma \cup \{-\})^2$, we define a score $\sigma(u, v)$. The score of an alignment \mathcal{A} is the sum of the scores of all pairs in the nodes of \mathcal{A} , that is, $\sum_{(u,v) \in \mathcal{A}} \sigma(u, v)$. The similarity $sim(F, G)$ of F and G is the score of optimal global alignment of F and G .

The local forest alignment problems focus on finding two local subforests of original forests such that they have optimal global alignment score. Here we define three problems of local forest alignment.

The *gapped forest alignment* problem is to find two gapped subforests F' and G' of F and G , respectively, such that the global alignment score $sim(F', G')$ is maximized.

$$lgsf(F, G) = \max\{sim(F', G') \mid F' \in gsf(F), G' \in gsf(G)\}$$

The *β -gapped forest alignment* problem is to find two β -gapped subforests F' and G' of F and G , respectively, maximizing the global alignment score $sim(F', G')$.

$$lgsf_\beta(F, G) = \max\{sim(F', G') \mid F' \in gsf_\beta(F), G' \in gsf_\beta(G)\}$$

The *closed forest alignment* problem is to find two closed subforests F' and G' of F and G , respectively, such that the global alignment score $sim(F', G')$ is maximized.

$$lcsf(F, G) = \max\{sim(F[u..u'], G[v..v']) \mid u..u' \in S(F), v..v' \in S(G)\}$$

1.2 Previous Results

RNAs can be modeled as annotated sequences [5] or labeled ordered trees. For comparing annotated sequences, a number of results have been done on global edit distance and global alignment [1, 3, 5, 6, 7, 10, 12, 13]. The only known local alignment result is given in [2].

This paper focuses on labeled ordered trees comparison. For global tree edit distance, results include [15, 19, 20]. The first algorithm for *global alignment* of labeled ordered trees was proposed by Jiang, Wang, and Zhang [11]. Their algorithm computes the optimal global alignment between two labeled ordered trees T_1 and T_2 in $O(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2))^2)$ time, where $|T_i|$ and $\deg(T_i)$ for $i \in \{1, 2\}$ denote the number of nodes in T_i and the degree of T_i , respectively. It was extended without affecting the asymptotic running time to the problem of optimally aligning two labeled ordered trees with gap penalties by Wang and Zhao [17]. In [17], Wang and Zhao also showed how to reduce the space complexity of the resulting algorithm from $O(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2)))$ to $O(\log(|T_1|) \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2)) \cdot \deg(T_1))$ at the expense of increasing the running time to $O(|T_1|^2 \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2))^2)$. A modification to the algorithm of Jiang *et al.* which yields a lower running time for *global alignment* of trees was given in [9].

As for computing $lcsf$ of labeled ordered trees, Höchsmann [8] gave an algorithm for $lcsf$ (they termed it as the $lcsf$). Backofen and Will [2] studied a problem that are called the $lgsf_\beta$ (this problem is equivalent to our $lgsf_\beta$, as we prove in Section 4). The following table summarizes the complexities of the previously most efficient algorithms for $lcsf$, $lgsf$, and $lgsf_\beta$.

Problem	Time complexity	Space complexity
$lcsf$ (See [8])	$O(F \cdot G \cdot \deg(F) \cdot \deg(G) \cdot (\deg(F) + \deg(G)))$	$O(F \cdot G \cdot \deg(F) \cdot \deg(G))$
$lgsf$	Not studied before	Not studied before
$lgsf_\beta$ (See [2])	$O(F ^2 \cdot G ^2 \cdot (F + G))$	$O(F \cdot G)$

1.3 Our Results and Organization of the Paper

In Section 2, we introduce some additional notations and derive a number of recursive formulas which form the basis of our main dynamic programming-based algorithm for solving $lgsf$, presented in Section 2.4. Next, in Sections 3.1 and 3.2 we refine our algorithm for $lgsf$ to solve $lgsf_\beta$ and $lcsf$ even more efficiently. In Section 4, we prove that $lgsf_\beta$ is equivalent to the local sequence-structure problem considered in [2] and describe practical applications of $lgsf$ related to finding structural motifs in RNA molecules. Finally, in Section 5, we discuss possible future extensions of our work.

The table below summarizes the complexities of our algorithms.

Problem	Time complexity	Space complexity
$lcsf$ (Section 3.2)	$O(F \cdot G \cdot (\deg(F) + \deg(G))^2)$	$O(F \cdot G \cdot (\deg(F) + \deg(G)))$
$lgsf$ (Section 2.4)	$O(F \cdot G \cdot \deg(F) \cdot \deg(G) \cdot (\deg(F) + \deg(G)))$	$O(F \cdot G \cdot \deg(F) \cdot \deg(G))$
$lgsf_\beta$ (Section 3.1)	$O(F \cdot G \cdot (\deg(F) + \deg(G))^2)$	$O(F \cdot G \cdot (\deg(F) + \deg(G)))$

2 The Local Gapped Subforest Alignment Problem

This section presents an algorithm to solve $lgsf$. To compute the similarity of two forests F and G through alignment, we consider the search space in a structurally recursive fashion. The following lemma acts as the base for our algorithm.

2.1 Base Lemma

Lemma 2. Let A be a forest, Σ a set of symbols, F, G forests over Σ . Then, $lgsf(A, F, G) = \max\{lgsf(A, F), lgsf(A, G)\}$.

$$\begin{array}{l}
\begin{array}{ccc}
u & v & \\
a & & \\
(-, v) & &
\end{array} & A & \begin{array}{ccc}
F & G & \\
(u, v) & (u, -) &
\end{array} \\
a = (u, v) & A[a_L..a_R] & F[u_L..u_R] \quad G[u_L..u_R] \\
A[r(a)..e(a)] & & F[r(u)..e(u)] \quad G[r(v)..e(v)] \\
a = (u, -) & v'' \in l(v)..e(v) & A[a_L..a_R] \\
F[u_L..u_R] & G[v..v''] & A[r(a)..e(a)] \quad F[r(u)..e(u)] \\
G[r(v'')..e(v)] & & \\
a = (-, v) & u'' \in l(u)..e(u) & A[a_L..a_R] \\
F[u..u''] & G[v_L..v_R] & A[r(a)..e(a)] \quad F[r(u'')..u'] \\
G[r(v)..e(v)] & &
\end{array}$$

2.2 Matrix Notations

From Lemma 1, we know that a gapped subforest at $F[u..u']$ can either be an α -gapped subforest at $F[u..u']$ or a β -gapped subforest at $F[u''..u']$, depending on its excluded interval in $u..u'$. Thus, given two Σ -labeled forests F and G , to find $lgsf(F, G)$, our algorithm computes 9 dynamic programming matrices depending on the types of the gapped subforests. For every $a, b \in \{\alpha, \beta, *\}$, we define the matrix D_{a-b} as follows:

Definition 4 (Matrix). $a, b \in \{\alpha, \beta, *\}$, $u..u' \in S(F)$, $v..v' \in S(G)$, $D_{a-b}[u..u'; v..v']$ is the maximum value of $sim(F', G')$ where $F' \in gsf_a(F[u..u'])$ and $G' \in gsf_b(G[v..v'])$.

$$D_{a-b}[u..u'; v..v'] = \max\{sim(F', G') \mid F' \in gsf_a(F[u..u']), G' \in gsf_b(G[v..v'])\}$$

2.3 Recursive Formulae

Given the above matrices, the local gapped subforest alignment score and the local β -gapped subforest alignment score of two forests F and G can be computed based on the following lemma.

Lemma 3. $F, G \in \Sigma$, $u..u' \in S(F)$, $v..v' \in S(G)$.

$$lgsf(F, G) = \max\{D_{*-}[u..u'; v..v'] \mid u..u' \in S(F), v..v' \in S(G)\}$$

$$lgsf_\beta(F, G) = \max\{D_{\beta-\beta}[u..u'; v..v'] \mid u..u' \in S(F), v..v' \in S(G)\}.$$

The next step is to derive recursive formulae. First, the general matrix D_{*-} is computed using the following lemma. The proof follows directly from Definitions 2 and 4 and together with Lemma 1, and is therefore omitted.

Lemma 4 (General Matrix).

$$D_{*-}[u..u'; v..v'] = \max \begin{cases} D_{\alpha-\alpha}[u..u'; v..v'] \\ \max_{v'' \in v..r(v')} \{D_{*-\beta}[u..u'; v''..v']\} \\ \max_{u'' \in u..r(u')} \{D_{\beta-*}[u''..u'; v..v']\} \end{cases}$$

Also from Definitions 2 and 4 and Lemma 1, we can straightforwardly derive the formulae to compute the matrices $D_{\alpha-*}, D_{*-\alpha}, D_{\beta-*}, D_{*-\beta}$ as follows:

Lemma 5 (Special Matrices). $D_{\alpha-*}, D_{*-\alpha}, D_{\beta-*}, D_{*-\beta}$

$$\begin{aligned}
 & D_{*-\beta} \\
 - D_{\alpha-*}[u..u'; v..v'] &= \max\{D_{\alpha-\alpha}[u..u'; v..v'], \max_{v'' \in v..r(v')} \{D_{\alpha-\beta}[u..u'; v''..v']\}\} \\
 - D_{*-\alpha}[u..u'; v..v'] &= \max\{D_{\alpha-\alpha}[u..u'; v..v'], \max_{u'' \in u..r(u')} \{D_{\beta-\alpha}[u''..u'; v..v']\}\} \\
 - D_{\beta-*}[u..u'; v..v'] &= \max\{D_{\beta-\alpha}[u..u'; v..v'], \max_{v'' \in v..r(v')} \{D_{\beta-\beta}[u..u'; v''..v']\}\} \\
 - D_{*-\beta}[u..u'; v..v'] &= \max\{D_{\alpha-\beta}[u..u'; v..v'], \max_{u'' \in u..r(u')} \{D_{\beta-\beta}[u''..u'; v..v']\}\}
 \end{aligned}$$

Now we proceed to the computations of $D_{\alpha-\alpha}$, $D_{\alpha-\beta}$, and $D_{\beta-\alpha}$. Because these formulae are more complicated, we provide a sketch of the proof.

Lemma 6 (Alpha-Alpha Matrix). $D_{\alpha-\alpha}[u..u'; v..v'] =$

$$\max \begin{cases} \sigma(u, v) + D_{*-*}[u_L..u_R; v_L..v_R] + D_{*-*}[r(u)..u'; r(v)..v'] \\ \sigma(u, -) + \max_{v'' \in l(v)..v'} \{D_{*-\beta}[u_L..u_R; v..v''] + D_{*-*}[r(u)..u'; r(v'')..v']\} \\ \sigma(u, -) + \max_{v'' \in l(v)..v'} \{D_{*-\alpha}[u_L..u_R; v..v''] + D_{*-\beta}[r(u)..u'; r(v'')..v']\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{D_{\beta-*}[u..u''; v_L..v_R] + D_{*-*}[r(u'')..u'; r(v)..v']\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{D_{\alpha-*}[u..u''; v_L..v_R] + D_{\beta-*}[r(u'')..u'; r(v)..v']\} \end{cases}$$

Let F' and G' be two α -gapped subforests at $F[u..u']$ and $G[v..v']$ such that their optimal global alignment A is optimal among those in $gsf_{\alpha}(F[u..u'])$ and $gsf_{\alpha}(G[v..v'])$. Let u^* , v^* , and a be the roots of the leftmost subtrees of F' , G' , and A respectively (i.e., $u^* = u$ and $v^* = v$, but when we refer to u^* and v^* we mean the roots in F' and G'). From Lemma 2, we consider 3 cases:

- Case 1: When $a = (u^*, v^*)$, by Lemma 2, $F'[u_L^*..u_R^*]$ is aligned with $G'[v_L^*..v_R^*]$, and $F'[r(u^*)..e(u^*)]$ is aligned with $G'[r(v^*)..e(v^*)]$. Since F' is an α -gapped subforest at $F[u..u']$, $F'[u_L^*..u_R^*]$ and $F'[r(u^*)..e(u^*)]$ are gapped subforests at $F[u_L..u_R]$ and $F[r(u)..u']$ respectively. Similarly, $G'[v_L^*..v_R^*]$ and $G'[r(v^*)..e(v^*)]$ are gapped subforests at $G[v_L..v_R]$ and $G[r(v)..v']$ respectively. Hence, the alignment score of A equals $\sigma(u, v) + A[u_L..u_R; v_L..v_R] + A[r(u)..u'; r(v)..v']$.
- Case 2: When $a = (u^*, -)$, by Lemma 2, there exists some $v'' \in l(v^*)..e(v^*)$ such that $F'[u_L^*..u_R^*]$ is aligned with $G'[v^*..v'']$, and $F'[r(u^*)..e(u^*)]$ is aligned with $G'[r(v'')..e(v^*)]$. Since F' is an α -gapped subforest at $F[u..u']$, $F'[u_L^*..u_R^*]$ and $F'[r(u^*)..e(u^*)]$ are gapped subforests at $F[u_L..u_R]$ and $F[r(u)..u']$ respectively. Besides, by definition, since G' is an α -gapped subforest of $G[v..v']$, G' allows exclusion of $G[x..y]$ from $G[v..v']$ for at most one sibling interval $x..y \subseteq v..v'$. Depending on whether or not $x..y \subseteq v..v''$, we have two subcases.
 - Case 2a: $x..y \subseteq v..v''$. Then $G'[v^*..v'']$ is a β -gapped subforest at $G[v..v'']$, and $G'[r(v'')..e(v^*)]$ is a gapped subforest at $G[r(v'')..v']$. Hence, the alignment score of A equals $\sigma(u, -) + \max_{v'' \in l(v)..v'} \{A_{*-\beta}[u_L..u_R; v..v''] + A[r(u)..u'; r(v'')..v']\}$.

- Case 2b: $x.y \not\subseteq v..v''$. Then $G'[v^*..v'']$ is an α -gapped subforest at $G[v..v'']$, and $G'[r(v'')..e(v^*)]$ is a β -gapped subforest at $G[r(v'')..v']$. Hence, the alignment score of A equals $\sigma(u, -) + \max_{v'' \in l(v)..v'} \{A_{*-\alpha}[u_L..u_R; v..v''] + A_{*-\beta}[r(u)..u'; r(v'')..v']\}$.

– Case 3: When $a = (-, v)$, the proof is symmetric to the proof for Case 2. From the above three cases, Lemma 6 thus follows. \square

In the same way as in the proof of Lemma 6, we can derive Lemmas 7 and 8 below for computing $D_{\alpha-\beta}[u..u', v..v']$, $D_{\beta-\alpha}[u..u', v..v']$, and $D_{\beta-\beta}[u..u', v..v']$.

Lemma 7 (Alpha-Beta matrix). $D_{\alpha-\beta}[u..u'; v..v'] =$

$$\max \begin{cases} \sigma(u, v) + D_{*-*}[u_L..u_R; v_L..v_R] + D_{*-\beta}[r(u)..u'; r(v)..v'] \\ \sigma(u, -) + \max_{v'' \in l(v)..v'} \{D_{*-\beta}[u_L..u_R; v..v''] + D_{*-\beta}[r(u)..u'; r(v'')..v']\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{D_{\beta-*}[u..u''; v_L..v_R] + D_{*-\beta}[r(u'')..u'; r(v)..v']\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{D_{\alpha-*}[u..u''; v_L..v_R] + D_{\beta-\beta}[r(u'')..u'; r(v)..v']\} \end{cases}$$

$$D_{\beta-\alpha}[u..u'; v..v']$$

Lemma 8 (Beta-Beta matrix). $D_{\beta-\beta}[u..u'; v..v'] =$

$$\max \begin{cases} \sigma(u, v) + D_{*-*}[u_L..u_R; v_L..v_R] + D_{\beta-\beta}[r(u)..u'; r(v)..v'] \\ \sigma(u, -) + \max_{v'' \in l(v)..v'} \{D_{*-\beta}[u_L..u_R; v..v''] + D_{\beta-\beta}[r(u)..u'; r(v'')..v']\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{D_{\beta-*}[u..u''; v_L..v_R] + D_{\beta-\beta}[r(u'')..u'; r(v)..v']\} \end{cases}$$

2.4 The Main Algorithm and Its Complexity

From the recursive formulae, we can derive the algorithm to compute $lgsf(F, G)$ straightforwardly. Basically, the algorithm computes $D_{a-b}[u..u'; v..v']$ for all $a, b \in \{\alpha, \beta, *\}$, $u..u' \in S(F)$, and $v..v' \in S(G)$. With the 9 matrices D_{a-b} for $a, b \in \{*, \alpha, \beta\}$, the optimal alignment can be easily found using a simple traceback. The complexity will remain the same. The lemma below states its time and space complexity.

Lemma 9. $lgsf(F, G) = O(|F||G|deg(F)deg(G))$
 $(deg(F) + deg(G)) = O(|F||G|deg(F)deg(G))$

3 Algorithms for Two Variants of the Local Gapped Subforest Alignment Problem

3.1 Local β -Gapped Subforest Alignment Problem

To improve the efficiency of the algorithm for the Local β -Gapped Subforest Alignment Problem, we construct a new matrix $B[u, v]$ as follows:

$$B[u; v] = \max\{D_{\beta-\beta}[u..u'; v..v'] \mid u..u' \in S(F), v..v' \in S(G)\}$$

From the definitions of $D_{\beta-\beta}[u..u'; v..v']$ and of matrix B, we have:

Lemma 10. $B[u; v] = \max\{sim(F', G') \mid F' \in gsf_\beta[u..u'], G' \in gsf_\beta[v..v']\}$

Together with Lemma 3, we also have:

Lemma 11. $lgsf_\beta(F, G) = \max\{B[u; v] \mid u \in F, v \in G\}$

The computation of the matrix B is formulated as:

Lemma 12. $B[u; v] = \max \begin{cases} \sigma(u, v) + D_{*-}[u_L..u_R; v_L..v_R] + \max\{B[r(u); r(v)], 0\} \\ \sigma(u, -) + \max_{v'' \in l(v)..v'} \{D_{*\beta}[u_L..u_R; v..v''] + \max\{B[r(u); r(v'')], 0\}\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{D_{\beta-*}[u..u''; v_L..v_R] + \max\{B[r(u''); r(v)], 0\}\} \end{cases}$

To compute $lgsf_\beta(F, G)$, we need to compute all the entries in $B[u, v]$. In this situation, we observe that it is unnecessary to compute all of the entries in $D_{a-b}[u..u'; v..v']$ for all $a, b \in \{\alpha, \beta, *\}$, $u..u' \in S(F)$ and $v..v' \in S(G)$. We just need to fill in, for all $a, b \in \{\alpha, \beta, *\}$, the entries of the form $D_{a-b}[u..u'; v..v']$ where $u' = e(u)$ or $v' = e(v)$. Thus, for all $a, b \in \{\alpha, \beta, *\}$, each matrix $D_{a-b}[u..u'; v..v']$ is divided into two sub-matrices: $D'_{a-b}[u; v..v']$ and $D''_{a-b}[u..u'; v]$, where $D'_{a-b}[u; v..v'] = D_{a-b}[u..e(u); v..v']$ and $D''_{a-b}[u..u'; v] = D_{a-b}[u..u'; v..e(v)]$.

Lemma 13. Local β -Gapped Subforest Alignment Problem $\dots O(|F||G|(deg(F) + deg(G))^2) \dots O(|F||G|(deg(F) + deg(G))) \dots$

3.2 Local Closed Subforest Alignment Problem

The \dots has been proposed and solved by [8] in $O(|F||G|deg(F)deg(G)(deg(F)+deg(G)))$ time and $O(|F||G|deg(F)deg(G))$ space. We propose a faster and more space-saving algorithm for this problem. Using the same technique as in Section 3.1, we construct a new matrix $B[u..v]$:

$$B[u; v] = \max\{sim(F[u..u'], G[v..v']) \mid u..u' \in S(F), v..v' \in S(G)\}$$

Therefore, we have the following lemma:

Lemma 14. $csf(F, G) = \max\{B[u; v] \mid u \in F, v \in G\}$

The computation of the matrix B is formulated as following:

Lemma 15. $B[u; v] = \max \begin{cases} \sigma(u, v) + GD[u_L..u_R; v_L..v_R] + \max\{B[r(u); r(v)], 0\} \\ \sigma(u, -) + \max_{v'' \in l(v)..v'} \{GD[u_L..u_R; v..v''] + \max\{B[r(u); r(v'')], 0\}\} \\ \sigma(-, v) + \max_{u'' \in l(u)..u'} \{GD[u..u''; v_L..v_R] + \max\{B[r(u''); r(v)], 0\}\} \end{cases}$

where $GD[u..u'; v..v']$ is the optimal global alignment score of two closed subforests $F[u..u']$ and $G[v..v']$. But we just need to fill in the entries $GD[u..u'; v..v']$ where either $u' = e(u)$ or $v' = e(v)$. These entries in the $GD[u..u'; v..v']$ matrix can be computed by [11] in $O(|F||G|(deg(F) + deg(G))^2)$ time. Therefore we have the following analysis:

Lemma 16. Local Closed Subforest Alignment Problem
 $O(|F||G|(deg(F) + deg(G))^2) \dots O(|F||G|(deg(F) + deg(G)))$

4 An Application to Find Local RNA Sequence-Structure Motifs

An RNA secondary structure is a combination of an RNA sequence and a set of base pairings called arcs, which are bound together by hydrogen bonds. An RNA secondary structure can be represented as an ordered forest (S, P) [1, 5, 6, 7, 10, 13], which is a tuple (S, P) where S is a sequence of bases $s_1s_2 \dots s_n$ and P is a set of arcs formed by the position pairs (i, j) 's. The majority of RNA secondary structures has the characteristic that no two arcs cross (i.e., there exists no two arcs (i, j) and (i', j') such that $i < i' < j < j'$). With this condition, it is also common to represent an RNA secondary structure as a labeled ordered forest F , where each node in F corresponds to a free base in the sequence (i.e., a base that does not pair with any other base) or an arc in the secondary structure such that:

- For any two nodes u and v in F , u is the parent of v iff u corresponds to the parent arc of v (i.e., the smallest arc enclosing the base/arc that v corresponds to).
- For any two nodes u and v in F , u is a right sibling of v iff u corresponds to a base/arc that lies to the right of the base/arc of v .

Biologists have noticed that two RNAs sharing similar local substructure (which is referred as motif) have similar functions. This observation motivates the problem of computing the maximum common local substructure of two RNAs. A number of ways to represent the local substructures of an RNA have been proposed. Among them, the local sequence-structure motif [8] is the most general one, because it can represent local RNA substructures whose bases are connected when represented as a motif graph. For example, the local sequence-structure motif can represent the putative SECIS-motif [18]. Formally, given an RNA represented by (S, P) , (S', P') is called a local sequence-structure motif [8] of (S, P) if and only if:

- S' is a subsequence of S , and P' is a subset of P induced from S' ;
- S' is arc-complete for (S, P) (i.e., for every $(i, j) \in P$, either $i, j \in S'$ or $i, j \notin S'$); and
- any interval $s_k \dots s_{k'}$ is called an exclusion of S' if $s_k \dots s_l \notin S'$ and $s_{k-1}, s_{l+1} \in S'$. Every exclusion $s_k \dots s_l$ of S' has an immediate successor, which is an arc $(i, j) \in P'$ such that $i < k < l < j$ and $j - i$ is minimized. Also, no two exclusions of S' share the same immediate successor.

Let F and F' be the forest representations of (S, P) and (S', P') . The lemma below shows that (S', P') is a local sequence-structure motif of (S, P) iff F' is a β -gapped subforest of F .

Lemma 17. (S', P') is a local sequence-structure motif of (S, P) iff F' is a β -gapped subforest of F .

(\Rightarrow) Suppose (S', P') is a local sequence-structure motif of (S, P) . First, every exclusion in S' should be arc-complete. Otherwise, there exists some arc connecting a base in S' and a base in the exclusion, thus it contradicts to the fact that S' is arc-complete. Hence, every exclusion is arc-complete and corresponds to a closed subforest $F[x_i..y_i]$. In other words, F' is formed by excluding some $F[x_i..y_i]$'s from some $F[u..v]$. Then, since no exclusion has the same immediate successor, we conclude that all $F[x_i..y_i]$'s do not share the same parent. Thus, F' should be a gapped subforest of F . Finally, as every exclusion has an immediate successor, there is no sibling interval $x_i..y_i$ that can be excluded at the root level of F' , because otherwise $F[x_i..y_i]$ would corresponds to an exclusion in S' with no immediate successor. Hence, F' is a β -gapped subforest of F .

(\Leftarrow) Suppose F' is a β -gapped subforest of F . F' is formed by excluding some closed subforests $F[x_i..y_i]$'s from a closed subforest $F[u..v]$. Hence, the corresponding S' is arc-complete. Since no sibling interval can be excluded at the root level of F' , an excluded sibling interval (if one exists) has a parent in F' . Thus the corresponding exclusion has an immediate successor. Lastly, since there is no $x_i..y_i$'s sharing the same parent, all exclusions have different immediate successors. Hence, (S', P') is a local sequence-structure motif of (S, P) . \square

Given Lemma 13 and the following lemma, the maximum local sequence-structure motif of two annotated sequences (S_1, P_1) and (S_2, P_2) can be computed in $O(|F||G|(deg(F) + deg(G))^2)$ time and $O(|F||G|deg(F)deg(G))$ space.

Lemma 18. *Let (S_1, P_1) and (S_2, P_2) be two annotated sequences. Let F_1 and F_2 be two closed subforests of (S_1, P_1) and (S_2, P_2) respectively. Then, the maximum local sequence-structure motif of (S_1, P_1) and (S_2, P_2) is the maximum local sequence-structure motif of (S_1, P_1) and (S_2, P_2) excluding F_1 and F_2 .*

5 Concluding Remarks

Our proposed problem and solutions motivate future development in local alignment of labeled ordered forests. One of the challenges is to find even more efficient algorithms for the local forest alignment problems. Any improvement can have a vital impact in RNA comparison and structure prediction applications. Another difficult task is to further generalize the local gapped subforest alignment problem by allowing exclusions of more than one closed subforest sharing the same parent. Lastly, new alignment models could be proposed to give more effective and efficient algorithms for RNA comparison and structure prediction problems.

References

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Computing the similarity of two sequences with nested arc annotations. *Theoretical Computer Science*, 312(2-3):337-358, 2004.
2. R. Backofen and S. Will. Local sequence-structure motifs in RNA. *Journal of Bioinformatics and Computational Biology*, to appear.

3. V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching* (CPM 95), volume 937 of *LNCS*, pages 1–16. Springer, 1995.
4. L. J. Collins, V. Moulton, and D. Penny. Use of RNA secondary structure for studying the evolution of RNase P and RNase MRP. *Journal of Molecular Evolution*, 51(3):194–204, 2000.
5. P. A. Evans. *Algorithms and Complexity for Annotated Sequence Analysis*. PhD thesis, University of Victoria, Canada, 1999.
6. P. A. Evans. Finding common subsequences with arcs and pseudoknots. In *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching* (CPM 99), volume 1645 of *LNCS*, pages 270–280. Springer, 1999.
7. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. In *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science* (FSTTCS 2002), volume 2556 of *LNCS*, pages 182–193. Springer, 2002.
8. M. Höchsmann, T. Töller, R. Giegerich, and S. Kurtz. Local similarity in RNA secondary structures. In *Proceedings of the Computational Systems Bioinformatics Conference* (CSB2003), pages 159–168, 2003.
9. J. Jansson and A. Lingas. A fast algorithm for optimal alignment between similar ordered trees. *Fundamenta Informaticae*, 56(1–2):105–120, 2003.
10. T. Jiang, G. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. *Journal of Discrete Algorithms*, 2(2):257–270, 2004.
11. T. Jiang, L. Wang, and K. Zhang. Alignment of trees – an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.
12. H.-P. Lenhof, K. Reinert, and M. Vingron. A polyhedral approach to RNA sequence structure alignment. In *Proceedings of the Annual International Conference on Computational Biology* (RECOMB 1998), pages 153–162, 1998.
13. G. Lin, Z.-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. *Journal of Computer and System Sciences*, 65(3):465–480, 2002.
14. B. A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6(4):309–318, 1990.
15. K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.
16. J. T. L. Wang and K. Zhang. Identifying consensus of trees through alignment. *Information Sciences*, 126(1–4):165–189, 2000.
17. L. Wang and J. Zhao. Parametric alignment of ordered trees. *Bioinformatics*, 19(17):2237–2245, 2003.
18. R. Wilting, S. Schorling, B. C. Persson, and A. Böck. Selenoprotein synthesis in archaea: Identification of an mRNA element of *Methanococcus jannaschii* probably directing selenocysteine insertion. *Journal of Molecular Biology*, 266(4):637–641, 1997.
19. K. Zhang and T. Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.
20. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

The Maximum Agreement of Two Nested Phylogenetic Networks

Jesper Jansson and Wing-Kin Sung

School of Computing, National University of Singapore, 3 Science Drive 2,
Singapore 117543

{jansson, ksung}@comp.nus.edu.sg

Abstract. Given a set \mathcal{N} of phylogenetic networks, the maximum agreement phylogenetic subnetwork problem (MASN) asks for a subnetwork contained in every $N_i \in \mathcal{N}$ with as many leaves as possible. MASN can be used to identify shared branching structure among phylogenetic networks or to measure their similarity. In this paper, we prove that the general case of MASN is NP-hard already for two phylogenetic networks, but that the problem can be solved efficiently if the two given phylogenetic networks exhibit a nested structure. We first show that the total number of nodes $|V(N)|$ in any nested phylogenetic network N with n leaves and nesting depth d is $O(n(d+1))$. We then describe an algorithm for testing if a given phylogenetic network is nested, and if so, determining its nesting depth in $O(|V(N)| \cdot (d+1))$ time. Next, we present a polynomial-time algorithm for MASN for two nested phylogenetic networks N_1, N_2 . Its running time is $O(|V(N_1)| \cdot |V(N_2)| \cdot (d_1+1) \cdot (d_2+1))$, where d_1 and d_2 denote the nesting depths of N_1 and N_2 , respectively. In contrast, the previously fastest algorithm for this problem runs in $O(|V(N_1)| \cdot |V(N_2)| \cdot 4^f)$ time, where $f \geq \max\{d_1, d_2\}$.

1 Introduction

Phylogenetic trees are commonly used to describe evolutionary relationships among a set of objects (e.g., biological species, proteins, viruses, or languages) produced by an evolutionary process, and can help scientists to understand the mechanisms of evolution as well as to classify the objects being studied and to organize information [15, 19]. However, evolutionary events such as horizontal gene transfer or hybrid speciation (often referred to as *reticulation events*) which suggest convergence between objects cannot be adequately represented in a single tree structure [3, 10, 11, 12, 16, 17, 18, 21]. Phylogenetic networks were introduced in order to solve this shortcoming by allowing internal nodes to have more than one parent so that each recombination event may be represented by a node with indegree greater than one. Various methods for constructing and comparing phylogenetic networks have been proposed recently [3, 4, 10, 12, 13, 16, 17, 18, 21].

Phylogenetic network *reconciliation* has many uses; one application described in [16] is to assess the topological accuracy of different phylogenetic network con-

struction methods¹. Another application for network comparison is to identify a subnetwork with as many leaves as possible which is contained in all of the networks in a given set (obtained, for example, by different construction methods or by using the same method on alternative data sets) to determine which ancestral relationships are present in all networks. Moreover, the size of such a subnetwork provides a measure of how similar the networks in a given set are. This problem was formalized as a computational problem called *Maximum Ancestral Set Network* (MASN) and initially studied in [4].

The general case of MASN is NP-hard for three or more phylogenetic networks [4]. In fact, it is NP-hard even for just two networks, as we prove in this paper. Fortunately, recombination events usually do not occur in an unrestricted manner [10, 21]. It is therefore important to know under what structural restrictions on the input networks the problem becomes efficiently solvable. Here, we investigate the computational complexity of MASN for two phylogenetic networks whose merge paths are *non-crossing*, which is a natural generalization of rooted, leaf-labeled trees and so called galled-trees previously studied in [10, 13, 17, 21] (see below for definitions), and prove that this case can be solved by a polynomial-time algorithm. The decomposition technique for nested phylogenetic networks we develop here may also be applicable to other computational and combinatorial problems related to phylogenetic network construction and comparison.

1.1 Problem Definition and Terminology

A *phylogenetic network* N is a connected, rooted, simple, directed acyclic graph in which: (1) each node has outdegree at most 2; (2) each node has indegree 1 or 2, except the root node which has indegree 0; (3) no node has both indegree 1 and outdegree 1; and (4) all nodes with outdegree 0 are labeled by elements from a finite set L in such a way that no two nodes are assigned the same label. From here on, nodes of outdegree 0 are referred to as *leaves* and identified with their corresponding elements in L . We denote the set of all nodes and the set of leaves in a phylogenetic network N by $V(N)$ and $\Lambda(N)$, respectively.

Given a phylogenetic network N and a set L' , a *restriction* of N to L' , denoted by $N \upharpoonright L'$, is defined as the phylogenetic network obtained by first deleting all nodes which are not on any directed path from the root to a leaf in L' along with their incident edges, and then, for every node with outdegree 1 and indegree less than 2, contracting its outgoing edge (any resulting set of multiple edges between two nodes is replaced by a single edge).

Given a set $\mathcal{N} = \{N_1, \dots, N_k\}$ of phylogenetic networks, an *ancestral set network* of \mathcal{N} is a phylogenetic network A such that $\Lambda(A) \subseteq \bigcap_{N_i \in \mathcal{N}} \Lambda(N_i)$ and for every $N_i \in \mathcal{N}$, A is isomorphic to a subgraph of $N_i \upharpoonright \Lambda(A)$ in which zero or more of the edges have been deleted and each outgoing edge from a node with

¹ To evaluate a construction method \mathcal{M} , repeat the following steps a number of times. First, randomly generate a network N and evolve a sequence down the edges of N according to some chosen model of evolution, then build a network N' for the resulting set of sequences using \mathcal{M} , and finally measure the similarity between N' and N .

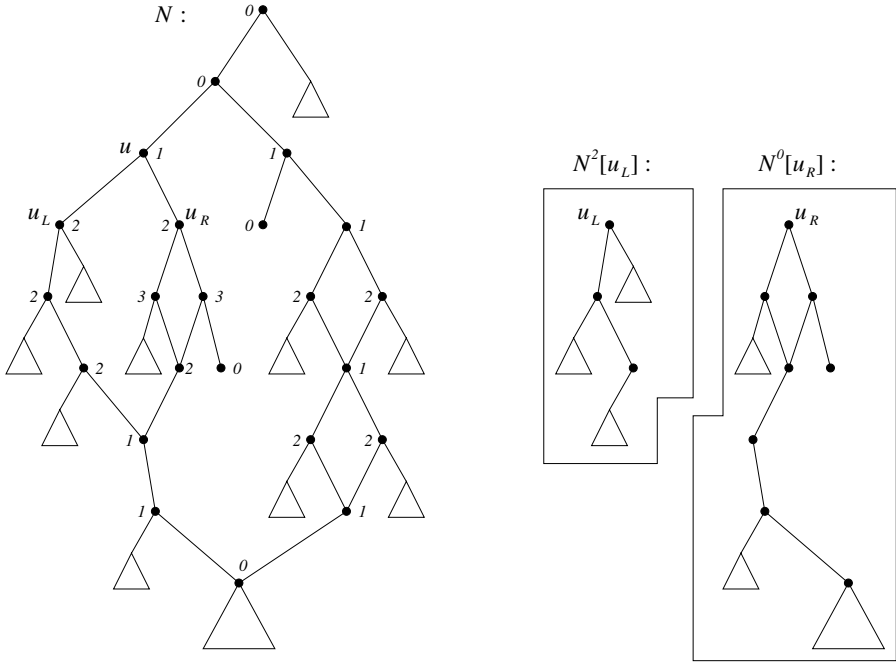


Fig. 1. N is a nested phylogenetic network with nesting depth 3 and u is a split node in N . The numbers shown next to the nodes of N are their respective nesting depths. $N^2[u_L]$ and $N^0[u_R]$ are the subgraphs of N displayed on the right

resulting outdegree 1 has been contracted. A
 \mathcal{N} is an agreement subnetwork of \mathcal{N} with the maximum possible number of leaves. (MASN)
 is: Given a set $\mathcal{N} = \{N_1, \dots, N_k\}$ of phylogenetic networks, find a maximum agreement subnetwork of \mathcal{N} . A leaf can appear in a maximum agreement subnetwork of \mathcal{N} only if it is present in every network in \mathcal{N} , so we assume without loss of generality that $A(N_1) = \dots = A(N_k)$ and call this leaf set L . Throughout this paper, we let n denote the number of different leaves and k the number of input networks, i.e., $n = |L|$ and $k = |\mathcal{N}|$ in the problem definition above.

To describe our results, we need the following terminology. Let N be a phylogenetic network. Recall that nodes with outdegree 0 are called We refer to nodes with indegree 2 as For any hybrid node h , every ancestor s of h such that h can be reached using two disjoint directed paths starting at the children of s is called a h . If s is a split node of h then any path starting at s and ending at h is a h , and any path starting at a child of s and ending at a parent of h is a h .

We say that N is a if for every two merge paths P_1, P_2 of two different hybrid nodes h_1, h_2 , either P_1 and P_2 are disjoint,

one is a subpath of the other, or their intersection equals either h_1 or h_2 . For each node u in a nested phylogenetic network N , define the *clipped merge path* of u , $d(u)$, as the number of hybrid nodes in N that have a clipped merge path passing through u . See Fig. 1 for an example. The *clipped merge depth* of N , denoted by $d(N)$, is the maximum value of $d(u)$ over all $u \in V(N)$. Note that if $d(N) = 0$ then N is a tree. Gusfield [10] defined a *galled tree* (also referred to in the literature as a *galled network* [17] or a *galled phylogenetic network* [21]) as a phylogenetic network in which all clipped merge paths are disjoint. For a discussion on the biological significance of galled-trees, see [10]. Clearly, $d(N) \leq 1$ if and only if N is a galled-tree. Thus, nested phylogenetic networks naturally extend the notion of rooted, leaf-labeled trees and galled-trees.

Finally, given any phylogenetic network N , let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is said to be a *f -galled tree* phylogenetic network if, for every biconnected component B in $\mathcal{U}(N)$, the subgraph of N induced by the set of nodes in B contains at most f nodes with indegree 2. If $f = 0$ then N is a tree, and we have $f = 1$ if and only if N is a nested phylogenetic network with nesting depth 1. If N is a nested phylogenetic network with nesting depth d then $f \geq d$.

1.2 Previous Results

Median-joining, split decomposition (SplitsTree), PYRAMIDS, statistical parsimony (TCS), molecular-variance parsimony (Arlequin), reticulogram (T-REX), and netting are some general methods for reconstructing phylogenetic networks (see [18] for a survey). More recently presented methods include NeighborNet [3] and Z-closure [12]. Algorithms for some reconstruction problems with additional constraints on the networks were given in [10, 13, 17, 21]; in particular, these papers considered problems involving constructing a network with nesting depth 1.

As for comparing two given networks, one method based on the Robinson-Foulds (RF) measure for phylogenetic trees was proposed in [16]. MASN was introduced in [4], where it was shown to be NP-hard if restricted to $k = 3$ and an $O(n^2)$ -time algorithm for the special case of two level-1 phylogenetic networks (i.e., having nesting depth 1) was presented. [4] also showed that MASN for two level- f networks N_1 and N_2 can be solved in $O(|V(N_1)| \cdot |V(N_2)| \cdot 4^f)$ time.

MASN generalizes a well-studied problem known as *Maximum Ancestral Set* (MAST)² (see, e.g., [1, 2, 5, 7, 9, 14, 20]) in which the input is a set of distinctly leaf-labeled trees and the goal is to compute a tree embedded in all of the input trees with the maximum possible number of labeled leaves. The fastest known algorithm for MAST for two trees runs in $O(\sqrt{D} n \log(2n/D))$ time, where n is the number of leaves and D is the maximum degree of the two input trees [14]. Note that this is $O(n \log n)$ for two trees with D bounded by a constant and $O(n^{1.5})$ for two trees with unbounded D . MAST is NP-hard for three trees with unbounded degrees [1], and solvable in $O(kn^3 + n^\delta)$ time for $k \geq 3$ trees, where δ is an upper bound on at least one of the input trees' degrees [2, 7].

² MAST is also known as *the maximum homeomorphic subtree problem* (MHT).

1.3 Our Results and Organization of Paper

In this paper, we focus on MASN for two nested phylogenetic networks. In Section 2, we derive some useful combinatorial properties of nested networks. We first prove that $|V(N)| = O(n(d+1))$ for any nested network N with n leaves and nesting depth d and then show how to test whether a given phylogenetic network is nested, and if so, determine its nesting depth in $O(|V(N)| \cdot (d+1))$ time. In Section 3, we present a simple and fast algorithm for solving MASN for two nested networks N_1 and N_2 running in $O(|V(N_1)| \cdot |V(N_2)| \cdot (d_1+1) \cdot (d_2+1))$ time, where d_1 and d_2 are the nesting depths of N_1 and N_2 , respectively. (The algorithm given in [4] could be applied here but its running time is $O(|V(N_1)| \cdot |V(N_2)| \cdot 4^f)$, where $f \geq \max\{d_1, d_2\}$.) For the special case $d_1 = 1, d_2 = 1$, i.e., two level-1 networks, the running time of our new algorithm coincides with the running time of $O(n^2)$ of the algorithm in [4]. Next, in Section 4, we strengthen the NP-hardness result of [4] by proving that MASN is NP-hard already for \mathcal{N} phylogenetic networks³. Finally, we discuss some open problems in Section 5. Proofs omitted due to space limitations will appear in the full-length version of this paper.

2 Preliminaries

We first state some basic properties of nested phylogenetic networks.

Lemma 1. *Let N be a nested phylogenetic network with nesting depth d . For any hybrid node h in N , there exists a unique split node s such that $d(h) = d(s)$.*

Because of Lemma 1, each hybrid node in a nested phylogenetic network corresponds to a unique split node. For any such hybrid node h and split node s , s is called the *split node of h* and h is called the *hybrid node of s* .

Lemma 2. *Let h be a hybrid node in a nested phylogenetic network with nesting depth d . Let s be the split node of h . Then $d(h) = d(s)$.*

We now derive an upper bound on the total number of nodes in a nested phylogenetic network. The next two lemmas generalize Lemma 3.2 in [4].

Lemma 3. *Let N be a nested phylogenetic network with nesting depth d and n leaves. Then $|V(N)| \leq n \cdot (d+1)$.*

Let $T_N(d)$ be the network N . For $i \in \{0, 1, \dots, d-1\}$, define $T_N(i)$ as the directed graph constructed from $T_N(i+1)$ as follows. For every hybrid node h in $T_N(i+1)$ with $d(h) = i$, remove h 's two incoming edges, contract the split node of h and all nodes on the two clipped merge paths of h to a single node s , and add

³ The reduction in [1] for proving the NP-hardness of MAST restricted to three trees with unbounded degrees cannot be used directly for MASN with $k = 2$ because it constructs *three* trees and because here we require all nodes to have outdegree at most two. Interestingly, MAST for two binary trees is solvable in $O(n \log n)$ time [5, 14].

a directed edge from s to h . $T_N(0)$ is a tree because every node with indegree 2 in N has indegree 1 in $T_N(0)$ and no contraction increases the indegree of any node. $T_N(0)$ has n leaves, so the number of internal nodes in $T_N(0)$ with outdegree > 1 is at most $n - 1$. Observe that at most d split nodes in N correspond to each internal node in $T_N(0)$ with outdegree > 1 and that the number of hybrid nodes in N equals the number of split nodes in N since N is nested. \square

Lemma 4. *If N is a nested phylogenetic network with n leaves and H hybrid nodes, then $|V(N)| \leq 2(n + H) - 1$.*

Let z_{ij} denote the number of nodes in N which have i incoming edges and j outgoing edges. By the definition of a phylogenetic network, the total number of nodes in N is $z_{02} + z_{10} + z_{12} + z_{20} + z_{21} + z_{22}$. For every $u \in V(N)$, let $in(u)$ and $out(u)$ denote the number of incoming and outgoing edges incident to u . Since

$$\begin{cases} \sum_{u \in V(N)} in(u) = z_{02} \cdot 0 + (z_{10} + z_{12}) \cdot 1 + (z_{20} + z_{21} + z_{22}) \cdot 2 \\ \sum_{u \in V(N)} out(u) = (z_{10} + z_{20}) \cdot 0 + z_{21} \cdot 1 + (z_{02} + z_{12} + z_{22}) \cdot 2 \end{cases}$$

and $\sum_{u \in V(N)} in(u) = \sum_{u \in V(N)} out(u)$, we have $z_{12} = z_{10} + 2z_{20} + z_{21} - 2z_{02}$. Next, $H = z_{20} + z_{21} + z_{22}$, $n = z_{10} + z_{20}$, and $z_{02} = 1$ give us $z_{12} \leq n + H - 2$. Hence, $|V(N)| \leq 1 + n + (n + H - 2) + H = 2n + 2H - 1$. \square

Theorem 1. *If N is a nested phylogenetic network with n leaves and d split nodes, then $|V(N)| = O(n(d + 1))$.*

Theorem 2. *If N is a nested phylogenetic network with n leaves and H hybrid nodes, then $|V(N)| = O(|V(N)| \cdot (H + 1))$ and $|V(N)| = O(|V(N)| \cdot (d(N) + 1))$.*

Use the following method to construct a list $L(u)$ for every $u \in V(N)$ consisting of all hybrid nodes which have a clipped merge path passing through u , plus u itself if u is a hybrid node. Associate an initially empty list $L(u)$ to each $u \in V(N)$, and define $L(\emptyset) = \emptyset$. Do a postorder traversal of the nodes of N . Whenever a non-leaf node u is visited, examine $L(u_L)$ and $L(u_R)$, where u_L and u_R are the children of u (if u only has one child then let u_R equal \emptyset). If $L(u_L)$ is empty then let $L(u) := L(u_R)$; else if $L(u_R)$ is empty then let $L(u) := L(u_L)$. Otherwise, check whether $L(u_L)$ equals $L(u_R)$. If no then N is not nested, and the algorithm terminates; if yes then let $L(u) := L(u_L)$ and remove the last element ℓ from $L(u)$ (here, u is in fact the split node for the hybrid node ℓ). Finally, if u is a hybrid node then insert u at the end of $L(u)$. Note that a node may be both a split node and a hybrid node. No $|L(u)|$ can exceed the number of hybrid nodes in N . Moreover, when the algorithm is finished, if N is a nested phylogenetic network then its nesting depth $d(N)$ equals the maximum length of $L(u)$ over all $u \in V(N)$ since $d(u) = |L(u)|$ for each non-hybrid node u . \square

3 An Algorithm for MASN for Two Nested Networks

In this section, we show how to solve MASN for two nested phylogenetic networks N_1, N_2 with n leaves in $O(|V(N_1)| \cdot |V(N_2)| \cdot (d_1 + 1) \cdot (d_2 + 1))$ time, where d_1 and d_2 are the nesting depths of N_1 and N_2 , respectively.

Let N be any nested phylogenetic network. From this point onward, assume that some arbitrary left-to-right ordering of the children of every node has been fixed. If $u \in V(N)$ has two children then let u_L and u_R denote the left and right child of u , respectively, and if u only has one child c then set $u_L = c$ and $u_R = \emptyset$. For every $u \in V(N)$, $N[u]$ is the subnetwork of N rooted at u , i.e., the minimal subgraph of N which includes all nodes and directed edges of N reachable from u . $N[\emptyset]$ refers to the empty network with no nodes or edges.

Each $u \in V(N)$ belongs to $d(u)$ different clipped merge paths. Since N is nested, the $d(u)$ different hybrid nodes corresponding to these clipped merge paths have nesting depths $0, 1, \dots, d(u) - 1$. For $i \in \{1, \dots, d(u)\}$, we define $h^i(u)$ as the hybrid node h which has a clipped merge path passing through u and which satisfies $d(h) = i - 1$. Next, for $i \in \{1, \dots, d(u)\}$, let $N^i[u]$ be the subgraph of $N[u]$ where $N[h^i(u)]$ and $h^i(u)$'s incoming edge have been removed, and let $N^0[u]$ be $N[u]$. Define $N^i[u]$ for $i > d(u)$ as $N^0[u]$ if u is not a hybrid node, and as $N[\emptyset]$ if u is a hybrid node. See Fig. 1 for an example. Intuitively, the parameter i informs us at which descendant hybrid node of u to cut $N[u]$ to obtain $N^i[u]$.

Lemma 5. $N, u \in V(N), 0 \leq j < i \leq d(u), N^i[u], N^j[u]$

Lemma 6. $N, u \in V(N), i \in \{0, 1, \dots, d(u)\}, N^i[u_L], N^x[u_R], N^x[u_L], N^i[u_R], x = d(u) + 1, u, x = i$

If u is a split node then let h be the hybrid node of s . By Lemma 2, $d(h) = d(u)$. Let c_1 be a child of u with $c_1 \neq h$ and let c_2 be the other child of u . We have $h^x(c_1) = h^{d(u)+1}(c_1) = h$, which means that $N^x[c_1]$ does not contain any nodes in $N[h]$; hence, $N^x[c_1]$ and $N^0[c_2]$ are disjoint, and Lemma 5 then implies that $N^x[c_1]$ and $N^i[c_2]$ are disjoint. Similarly, $N^i[c_1]$ and $N^x[c_2]$ are disjoint (if $c_2 \neq h$ then $h^x(c_2) = h^{d(u)+1}(c_2) = h$ so $N^x[c_2]$ contains no nodes in $N[h]$ and thus no nodes in $N^i[c_1]$; if $c_2 = h$ then $N^x[c_2] = N^{d(u)+1}[h] = N^{d(h)+1}[h] = N[\emptyset]$).

If u is not a split node then $N[u_L]$ ($= N^0[u_L]$) and $N[u_R]$ ($= N^0[u_R]$) are always disjoint. By Lemma 5, $N^i[u_L]$ and $N^i[u_R]$ are disjoint. \square

For any two phylogenetic networks N_1, N_2 , define $Masn(N_1, N_2)$ as the number of leaves in a maximum agreement subnetwork. If N_1 or N_2 is an empty network then $Masn(N_1, N_2)$ is 0. Otherwise, $Masn(N_1, N_2)$ for two nested networks can be expressed recursively using the following lemma which is a generalization of the main lemma in [20] for MAST. In the *Match* case, when trying to match two subnetworks $N_1^i[u_L]$ and $N_1^x[u_R]$ to two subnetworks $N_2^k[v_L]$ and $N_2^y[v_R]$, Lemma 6 ensures that the set of nodes in the intersection of $V(N_1[u_L])$ and $V(N_1[u_R])$ is matched to only one of $N_2^k[v_L]$ and $N_2^y[v_R]$, and vice versa.

Lemma 7. Let N_1 and N_2 be two phylogenetic networks with leaf sets $V(N_1)$ and $V(N_2)$, respectively. Let $u, v \in V(N_1) \times V(N_2)$ and $0 \leq i \leq d(u)$, $0 \leq k \leq d(v)$.

$$Masn(N_1^i[u], N_2^k[v]) = \begin{cases} |\Lambda(N_1^i[u]) \cap \Lambda(N_2^k[v])|, & \text{if at least one of } u \text{ and } v \\ & \text{is a leaf} \\ \max\{Diag(N_1^i[u], N_2^k[v]), Match(N_1^i[u], N_2^k[v])\}, & \text{otherwise} \end{cases}$$

$$Diag(N_1^i[u], N_2^k[v]) = \max\{Masn(N_1^i[u], N_2^k[v_L]), Masn(N_1^i[u], N_2^k[v_R]), Masn(N_1^i[u_L], N_2^k[v]), Masn(N_1^i[u_R], N_2^k[v])\}$$

$$Match(N_1^i[u], N_2^k[v]) = \max\{Masn(N_1^i[u_L], N_2^k[v_L]) + Masn(N_1^x[u_R], N_2^y[v_R]), Masn(N_1^i[u_L], N_2^y[v_L]) + Masn(N_1^x[u_R], N_2^k[v_R]), Masn(N_1^i[u_L], N_2^k[v_R]) + Masn(N_1^x[u_R], N_2^y[v_L]), Masn(N_1^i[u_L], N_2^y[v_R]) + Masn(N_1^x[u_R], N_2^k[v_L]), Masn(N_1^x[u_L], N_2^k[v_L]) + Masn(N_1^i[u_R], N_2^y[v_R]), Masn(N_1^x[u_L], N_2^y[v_L]) + Masn(N_1^i[u_R], N_2^k[v_R]), Masn(N_1^x[u_L], N_2^k[v_R]) + Masn(N_1^i[u_R], N_2^y[v_L]), Masn(N_1^x[u_L], N_2^y[v_R]) + Masn(N_1^i[u_R], N_2^k[v_L])\}$$

$$x = \begin{cases} d(u) + 1, & \text{if } u \text{ is a split node} \\ i, & \text{otherwise} \end{cases}$$

$$y = \begin{cases} d(v) + 1, & \text{if } v \text{ is a split node} \\ k, & \text{otherwise} \end{cases}$$

Now, given two nested phylogenetic networks N_1 and N_2 , we can use Lemma 7 to compute $Masn(N_1^i[u], N_2^k[v])$ for all $0 \leq i \leq d(u)$ and $0 \leq k \leq d(v)$ by applying dynamic programming in a bottom-up manner. The resulting algorithm (Algorithm 1) *Masn* is listed in Fig. 2.

Lemma 8. Let N_1 and N_2 be two phylogenetic networks with leaf sets $V(N_1)$ and $V(N_2)$, respectively. Then the time complexity of *Masn* is $O(|V(N_1)| \cdot |V(N_2)| \cdot (d(N_1) + 1) \cdot (d(N_2) + 1))$.

Algorithm 1 *Masn* can be modified to compute the set of leaves in a maximum agreement subnetwork without increasing the asymptotic running time by also recording information about how each *Masn*-value is attained as it is computed, e.g., by saving pointers. To construct an actual maximum agreement subnetwork from such a set L' , we may use a standard traceback technique to obtain a tree with leaf set L' which is an agreement subnetwork. This yields:

Theorem 3. Let N_1 and N_2 be two phylogenetic networks with leaf sets $V(N_1)$ and $V(N_2)$, respectively. Let d_1 and d_2 be the maximum depths of N_1 and N_2 , respectively. Then the time complexity of the algorithm for constructing a maximum agreement subnetwork is $O(|V(N_1)| \cdot |V(N_2)| \cdot (d_1 + 1) \cdot (d_2 + 1))$.

Algorithm *NestedMasn***Input:** Two nested phylogenetic networks N_1 and N_2 .**Output:** The number of leaves in a maximum agreement subnetwork of $\{N_1, N_2\}$.

- 1 Compute and store $d(u)$ and $h^i(u)$ for all $u \in V(N_1) \cup V(N_2)$, $i \in \{1, \dots, d(u)\}$.
 - 2 Let \mathcal{O} be the lexicographic ordering of $V(N_1) \times V(N_2)$, where the nodes in each $V(N_i)$ are ordered according to postorder.
 - 3 **for** each $(u, v) \in V(N_1) \times V(N_2)$ in increasing order in \mathcal{O} **do**
 Compute $Masn(N_1^i[u], N_2^k[v])$ for all $0 \leq i \leq d(u)$, $0 \leq k \leq d(v)$ by using the expression in Lemma 7.
endfor
 - 4 **return** $Masn(N_1^0[r_1], N_2^0[r_2])$, where r_i is the root of N_i for $i \in \{1, 2\}$.
- End** *NestedMasn*

Fig. 2. A dynamic programming algorithm for computing all values of *Masn*

4 MASN with $k = 2$ Is NP-Hard

To prove the NP-hardness of MASN for every fixed $k \geq 2$, we provide a polynomial-time reduction from the following problem.

Three-Dimensional Matching (3DM): Given a set $M \subseteq X \times Y \times Z$, where X , Y , and Z are disjoint sets and $X = \{x_1, \dots, x_q\}$, $Y = \{y_1, \dots, y_q\}$, $Z = \{z_1, \dots, z_q\}$, is there a subset M' of M with $|M'| = q$ such that M' is a matching, i.e., such that for every pair $e, f \in M'$ it holds that e and f differ in all coordinates?

3DM is NP-complete (see, e.g., [8]). Given an arbitrary instance of 3DM, construct an instance of MASN with two phylogenetic networks N_1 and N_2 with a leaf set L as described below. The elements of M are encoded in subtrees called S_{x_i, z_k} in N_1 and in subtrees called U_{y_j} in N_2 . The purpose of the subtrees named A_{x_i} , B_{x_i, z_k} , and W_{z_k} is to make sure that for any two triples e and f in M , a maximum agreement subnetwork of N_1 and N_2 can contain both of the two leaves representing e and f if and only if e and f differ in all coordinates.

Take $L = M \cup A \cup B$, where A is a set of $q^6 \cdot (q + 2)$ elements not in M and B is a set of q^6 elements not in M or A . Let $A_{x_0}, \dots, A_{x_q}, A_{x_{q+1}}$ be $q + 2$ binary trees with q^6 leaves each, distinctly labeled by A . For every $(x_i, z_k) \in X \times Z$, let B_{x_i, z_k} be a binary tree with q^4 leaves, distinctly labeled by B . For every $(x_i, z_k) \in X \times Z$, define: (1) M_{x_i, z_k} as the subset of M containing all triples of the form (x_i, y, z_k) where $y \in Y$; and (2) S_{x_i, z_k} to be a tree obtained from a binary caterpillar tree with $|M_{x_i, z_k}| + 1$ leaves distinctly labeled by M_{x_i, z_k} and where one of the bottommost leaves has been replaced by the root of B_{x_i, z_k} . See Fig. 3. For every $y_j \in Y$, define: (1) M_{y_j} as the subset of M containing all triples of the form (x, y_j, z) where $x \in X$ and $z \in Z$; and (2) U_{y_j} to be a binary caterpillar tree with $|M_{y_j}| + q$ leaves in which the $|M_{y_j}|$ leaves closest to the root are distinctly labeled by M_{y_j} and the rest are unlabeled nodes referred to as v_{y_j, z_k} for $1 \leq k \leq q$. Then, for

every $z_k \in Z$, define W_{z_k} to be a tree obtained from the binary caterpillar tree with q leaves by replacing the leaves with the roots of $B_{x_1, z_k}, \dots, B_{x_q, z_k}$.

Next, let P be any sorting network (see, e.g., [6]) for q elements with a polynomial number p of comparator stages. Build a directed acyclic graph Q from P with $(p + 1) \cdot q$ nodes $\{Q_{i,j} \mid 1 \leq i \leq p+1, 1 \leq j \leq q\}$ such that there is a directed edge $(Q_{i,j}, Q_{i+1,j})$ for every $1 \leq i \leq p$ and $1 \leq j \leq q$, and two directed edges $(Q_{i,j}, Q_{i+1,k})$ and $(Q_{i,k}, Q_{i+1,j})$ for every comparator (j, k) at stage i in P for $1 \leq i \leq p$, as illustrated in Fig. 4. Furthermore, construct q directed paths $\{G_1, \dots, G_q\}$ where each $G_k = (G_{1,k}, \dots, G_{q,k})$.

Let N_1 be a phylogenetic network (in fact, a leaf-labeled binary tree) obtained by attaching to a directed path $(m_1, m_2, \dots, m_{q^2+q+2})$, in order of non-decreasing distance from m_1 , the roots of $A_{x_0}, S_{x_1, z_1}, S_{x_1, z_2}, \dots, S_{x_1, z_q}, A_{x_1}, S_{x_2, z_1}, \dots, S_{x_q, z_q}, A_{x_q}$, and $A_{x_{q+1}}$, and letting m_1 be the root of N_1 . See Fig. 5. The phylogenetic network N_2 is obtained by first attaching to a directed path $(n_1, n_2, \dots, n_{2q+2})$, in order of non-decreasing distance from n_1 , the root of A_{x_0} , the node $Q_{1,1}$, the root of A_{x_1} , the node $Q_{1,2}$, the root of A_{x_2}, \dots , the root of A_{x_q} , and the root of $A_{x_{q+1}}$, and letting n_1 be the root of N_2 . Then, for $j \in \{1, \dots, q\}$, let $Q_{p+1,j}$ coincide with the root of U_{y_j} , and for every $1 \leq j \leq q$ and $1 \leq k \leq q$ add a directed edge $(v_{y_j, z_k}, G_{j,k})$. Next, for every $1 \leq k \leq q$ add a directed edge from $G_{q,k}$ to the root of W_{z_k} . Finally, for every node in N_1 and N_2 having indegree 1 and outdegree 1, contract its outgoing edge.

Lemma 9. $(N_1, N_2) \prec \dots \prec q^7 + 2q^6 + q^5 + q$. $M \prec \dots \prec q$

Theorem 4. $k = 2$

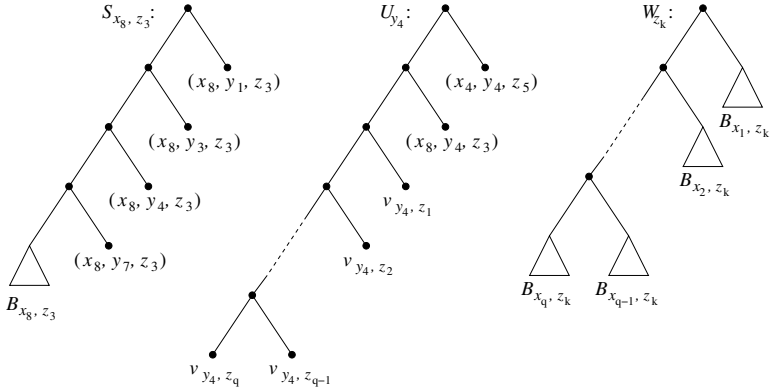


Fig. 3. Assume $M_{x_8, z_3} = \{(x_8, y_1, z_3), (x_8, y_3, z_3), (x_8, y_4, z_3), (x_8, y_7, z_3)\}$ and $M_{y_4} = \{(x_4, y_4, z_5), (x_8, y_4, z_3)\}$. S_{x_8, z_3} and U_{y_4} are shown on the left and in the center, respectively. The structure of each W_{z_k} is shown on the right

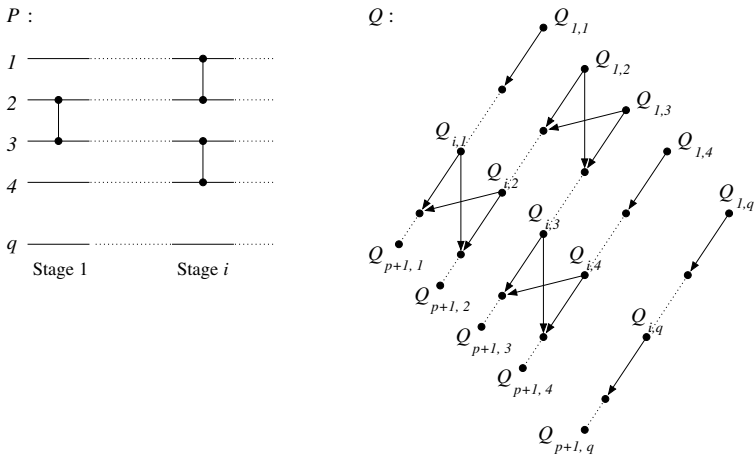


Fig. 4. The sorting network P on the left yields a directed acyclic graph Q

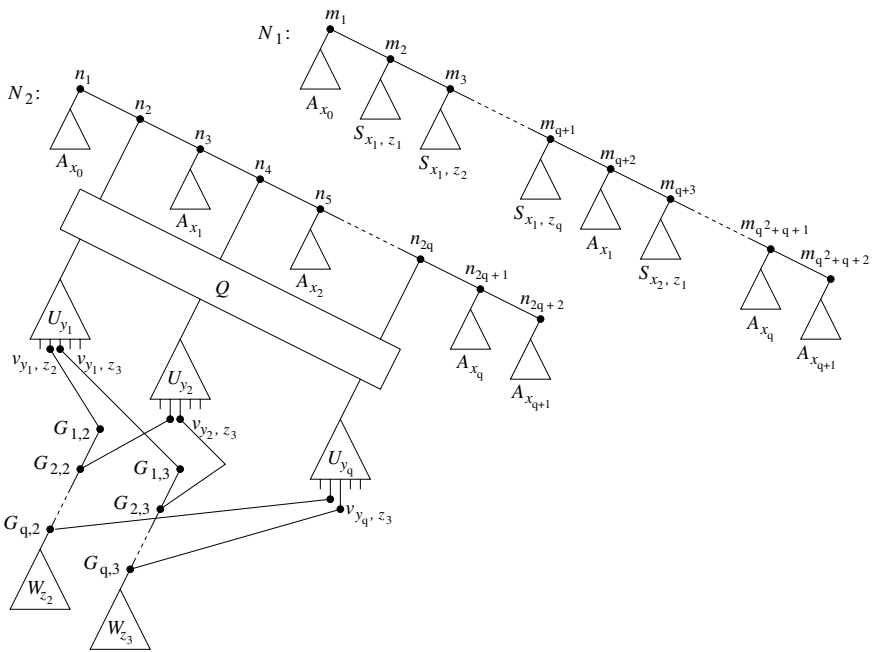


Fig. 5. The phylogenetic networks N_1 and N_2

5 Open Problems

Does MASN for other types of structurally restricted phylogenetic networks admit efficient algorithms? In particular, is it possible to extend our method in

Section 3 to two networks in which every hybrid node has exactly one split node? An example of such a network is shown in Fig. 6. It would also be interesting to investigate if any other problems which are hard to solve for unrestricted phylogenetic networks but solvable in polynomial time for galled-trees (i.e., networks with nesting depth 1)⁴ can be solved efficiently for nested phylogenetic networks.

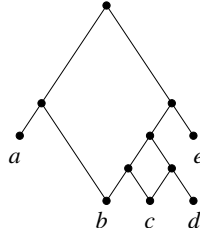


Fig. 6. This network is not nested, yet every hybrid node has exactly one split node and every split node has exactly one hybrid node, i.e., the converse of Lemma 1 is not true

We believe MASN for more than two nested phylogenetic networks can be solved in polynomial time when $k = O(1)$. On the other hand, if the outdegree 2 constraint in the definition of phylogenetic networks is removed, MASN seems to be NP-hard already for two networks with nesting depth 1. The final open question is: can the running time of our algorithm for two nested phylogenetic networks be improved, e.g., by applying sparsification techniques?

References

1. A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM J. on Computing*, 26:1656–1669, 1997.
2. D. Bryant. *Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis*. PhD thesis, Univ. of Canterbury, New Zealand, 1997.
3. D. Bryant and V. Moulton. NeighborNet: an agglomerative method for the construction of planar phylogenetic networks. In *Proc. of the 2nd Workshop on Algorithms in Bioinformatics (WABI 2002)*, volume 2452 of *LNCS*, pages 375–391. Springer, 2002.
4. C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung. Computing the maximum agreement of phylogenetic networks. In *Proc. of Computing: the 10th Australasian Theory Symposium (CATS 2004)*, volume 91 of *ENTCS*, pages 134–147. Elsevier, 2004.
5. R. Cole, M. Farach-Colton, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM J. on Computing*, 30(5):1385–1404, 2000.
6. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. MIT Press, 1990.
7. M. Farach, T. Przytycka, and M. Thorup. On the agreement of many trees. *Information Processing Letters*, 55:297–301, 1995.

⁴ For example, the perfect phylogenetic network with recombination problem is NP-hard for unrestricted networks [21], but polynomial-time solvable for galled-trees [10].

8. M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.
9. L. Gašieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, 3:183–197, 1999.
10. D. Gusfield, S. Eddhu, and C. Langley. Efficient reconstruction of phylogenetic networks with constrained recombination. In *Proc. of the Computational Systems Bioinformatics Conference (CSB2003)*, pages 363–374, 2003.
11. J. Hein. Reconstructing evolution of sequences subject to recombination using parsimony. *Mathematical Biosciences*, 98(2):185–200, 1990.
12. D. H. Huson, T. Dezulian, T. Klöpper, and M. Steel. Phylogenetic super-networks from partial trees. In *Proc. of the 4th Workshop on Algorithms in Bioinformatics (WABI 2004)*, to appear.
13. J. Jansson and W.-K. Sung. Inferring a level-1 phylogenetic network from a dense set of rooted triplets. In *Proc. of the 10th International Computing and Combinatorics Conference (COCOON 2004)*, to appear.
14. M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212–233, 2001.
15. W.-H. Li. *Molecular Evolution*. Sinauer Associates, Inc., Sunderland, 1997.
16. L. Nakhleh, J. Sun, T. Warnow, C. R. Linder, B. M. E. Moret, and A. Tholse. Towards the development of computational tools for evaluating phylogenetic reconstruction methods. In *Proc. of the 8th Pacific Symposium on Biocomputing (PSB 2003)*, pages 315–326, 2003.
17. L. Nakhleh, T. Warnow, and C. R. Linder. Reconstructing reticulate evolution in species – theory and practice. In *Proc. of the 8th Annual International Conf. on Research in Computational Molecular Biology (RECOMB 2004)*, pages 337–346, 2004.
18. D. Posada and K. A. Crandall. Intraspecific gene genealogies: trees grafting into networks. *TRENDS in Ecology & Evolution*, 16(1):37–45, 2001.
19. J. Setubal and J. Meidanis. *Introduction to Comp. Molecular Biology*. PWS, 1997.
20. M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48:77–82, 1993.
21. L. Wang, K. Zhang, and L. Zhang. Perfect phylogenetic networks with recombination. *Journal of Computational Biology*, 8(1):69–78, 2001.

Sequences of Radius k : How to Fetch Many Huge Objects into Small Memory for Pairwise Computations

Jerzy W. Jaromczyk^{1,**} and Zbigniew Lonc²

¹ University of Kentucky, Lexington, KY, USA
jurek@cs.uky.edu

² Warsaw University of Technology, Warsaw, Poland
zblonc@mini.pw.edu.pl

Abstract. Let a_1, a_2, \dots, a_m be a sequence over $[n] = \{1, \dots, n\}$. We say that a sequence a_1, a_2, \dots, a_m has the k -radius property if every pair of different elements in $[n]$ occurs at least once within distance at most k ; the distance $d(a_i, a_j) = |i - j|$. We demonstrate lower and (asymptotically) matching upper bounds for sequences with the k -radius property. Such sequences are applicable, for example, in computations of two-argument functions for all $\binom{n}{2}$ pairs of large objects such as medical images, bitmaps or matrices, when processing occurs in a memory of size capable of storing $k + 1$ objects, $k < n$. We focus on the model when elements are read into the memory in a FIFO fashion that correspond to streaming the data or a special type of caching. We present asymptotically optimal constructions; they are based on the Euler totient theorem and recursion.

1 Introduction

The problem that we study originated in the context of computing a two-argument function (which we denote by g) for all pairs of n large objects, such as medical images, bitmaps or matrices [GJ02]. The memory size is too small to store all the objects at once; we assume it can store at most $k + 1$ objects at the same time. For that reason, the simple two-loop algorithm that iterates through the pairs of objects is not useful, as most of the objects are not readily available. The task is to provide the shortest possible sequence of operations that will ensure that, for all pairs (i, j) , at some point in time both o_i and o_j will reside in memory and $g(o_i, o_j)$ (as well as $g(o_j, o_i)$, if g is non-symmetric) can be evaluated.

The operation assumes that, if memory is full, the next element takes the place of one of the elements currently residing in memory. The particular

** This work was supported in part by the University of Kentucky subcontract of grants 5P20RR016481-03 and 2P20RR016481-04 from NCRR-NIH, and by KY NSF EPSCOR grant EPS-0132295.

selection of the element to be replaced is a part of the strategy. The replacement strategy based on a FIFO queue of size $k + 1$ (that is, remove the first element and append a new element to the end of the sequence) is particularly interesting because of its applications. For example, it appears in processing a large number of huge objects located in a remote database when locally storing fetched data may be either impossible or impractical and where limited bandwidth and time necessitate efficient scheduling of the data requests. Moreover, FIFO-like processing allows for read-ahead requests to preemptively fetch data. There are other applications of k -radius sequences; e.g., they may be viewed as a systematic and efficient scheduling for the caching process [SChD02].

The problem can be formalized as follows.

Let a_1, a_2, \dots, a_m be a sequence over $[n]$. The distance $d(a_i, a_j) = |i - j|$. We say that a sequence a_1, a_2, \dots, a_m has the k -radius property if every pair of different elements in $[n]$ occurs at least once within distance at most k . In other words, each pair of objects will appear at least once inside a window of size $k + 1$ that slides along the sequence.

In $\boxed{1} \boxed{2} \boxed{3} \boxed{4} \boxed{5} \boxed{6} \boxed{1} \boxed{2} \boxed{4} \boxed{5} \boxed{3} \boxed{6}$ of length 12, each of 15 pairs of numbers from $\{1, 2, 3, 4, 5, 6\}$ occurs within a distance of at most 2. This 2-radius sequence for $n = 6$ is a shortest (although not unique) such sequence, as we will see later. Some objects, such as $\boxed{1}$ and $\boxed{2}$ appear in this sequence within distance not larger than 2 more than once. We place numbers in boxes to indicate that each element in the sequence consists of potentially huge satellite data.

We study the following questions:

Question 1. What is the least length $f_k(n)$ of a sequence over $[n] = \{1, 2, \dots, n\}$ that has the k -radius property?

Question 2. How can we construct short sequences with the k -radius property?

We prove that

$$f_1(n) = \frac{1}{2}n^2(1 + o(1)), \quad f_2(n) = \frac{1}{4}n^2(1 + o(1)),$$

and

$$\frac{1}{2k}n^2(1 + o(1)) \leq f_k(n) \leq \frac{1}{4 \cdot \lfloor (k+1)/3 \rfloor}n^2(1 + o(1))$$

(for a fixed $k > 2$). We also show a construction of 2-radius sequences and use it as a ground condition in a recursive algorithm for constructing asymptotically optimal sequences with the k -radius property for $k > 2$. The correctness of the construction is based on the Euler totient theorem for congruences (mod n) of powers of the totient function $\phi(n)$: the number of numbers relatively prime with n [GKP89].

2 Simple Cases, Lower Bounds and Examples

A result for all pairs appearing consecutively in a sequence (in our terminology, the 1-radius property) was obtained by Ghosh [G75]; see also [LTT81]. It was formulated in the context of databases and the consecutive 1 property.

Theorem 1. ([G75], [LTT81])

$$f_1(n) = \begin{cases} \binom{n}{2} + 1, & \text{if } n \leq 3 \\ \binom{n}{2} + \frac{1}{2}n, & \text{if } n \geq 4 \end{cases}$$

For n objects, the length of the shortest sequence with the consecutive 1 property depends on k and its length is bounded from below as follows:

Theorem 2.

$$f_k(n) \geq \left\lceil \frac{1}{k} \binom{n}{2} + \frac{k+1}{2} \right\rceil$$

There are $m-i, i = 1, 2, \dots, k$, pairs of objects in a sequence a_1, a_2, \dots, a_m for which the distance is i . Hence we have

$$\sum_{i=1}^k (m-i) = mk - (1 + 2 + \dots + k) = mk - \frac{(k+1)k}{2}$$

pairs for which the distance is at most k . Thus, if the sequence a_1, a_2, \dots, a_m contains all pairs over $[n]$ within distance at most k , then

$$\begin{aligned} \binom{n}{2} &\leq mk - \frac{(k+1)k}{2} \\ f_k(m) \geq m &\geq \frac{1}{k} \binom{n}{2} + \frac{k+1}{2}. \end{aligned}$$

■

The above lower bound can be slightly improved.

Theorem 3.

$$f_k(n) \geq \left\lceil \frac{n-1}{2k} \right\rceil n + \sum_{j=1}^k \left(\left\lceil \frac{n+k-j}{2k} \right\rceil - \left\lceil \frac{n-1}{2k} \right\rceil \right)$$

Let a_1, a_2, \dots, a_m be a sequence over $[n]$ with the k -radius property and $m = f_k(n)$ (i.e. the sequence is of the shortest possible length). We claim that the first k objects a_1, a_2, \dots, a_k are pairwise different. Indeed, if $a_i = a_j, 1 \leq i < j \leq k$, then we delete a_i . The resulting sequence still has the k -radius property, in contradiction with the definition of m .

Let $r(i)$ be the number of occurrences of i in the sequence a_1, a_2, \dots, a_m , $i = 1, 2, \dots, n$. Obviously, for every $i \in [n]$, $2kr(i) \geq n - 1$, so $r(i) \geq \lceil \frac{n-1}{2k} \rceil$.

For $j = 1, \dots, k$, object a_j has only $j - 1 + k$ objects within distance at most k . Hence $j - 1 + k + (r(a_j) - 1) \cdot 2k \geq n - 1$

so

$$r(a_j) \geq \left\lceil \frac{n+k-j}{2k} \right\rceil, \quad j = 1, 2, \dots, k$$

As a_1, a_2, \dots, a_k are pairwise different,

$$\begin{aligned} f_k(n) = m &= \sum_{i=1}^n r(i) \geq (n-k) \left\lceil \frac{n-1}{2k} \right\rceil + \sum_{j=1}^k \left\lceil \frac{n+k-j}{2k} \right\rceil \\ &= \left\lceil \frac{n-1}{2k} \right\rceil n + \sum_{j=1}^k \left(\left\lceil \frac{n+k-j}{2k} \right\rceil - \left\lceil \frac{n-1}{2k} \right\rceil \right). \end{aligned}$$

■

Corollary 1.

$$f_2(n) \geq \begin{cases} \frac{1}{2} \binom{n}{2} + \frac{1}{4}n + 1, & n \equiv 0 \pmod{4} \\ \frac{1}{2} \binom{n}{2} + 2, & n \equiv 1 \pmod{4} \\ \frac{1}{2} \binom{n}{2} + \frac{3}{4}n, & n \equiv 2 \pmod{4} \\ \frac{1}{2} \binom{n}{2} + \frac{1}{2}n, & n \equiv 3 \pmod{4} \end{cases}$$

The bound given by Theorem 3 is for $k = 2$ not worse than the bound in Theorem 2.

For small values of n we can directly compare the lower and upper bounds.

$f_2(2) = 2$	12
$f_2(3) = 3$	123
$f_2(4) = 5$	12341
$f_2(5) = 7$	1234512
$f_2(6) = 12$	123456124536
$f_2(7) = 14$	12345632756147
$f_2(8) \geq 17$	

The lower bounds follow from the Corollary 1.

3 Construction for $k = 2$

The presented construction of a sequence with the $\text{rad}(a_i) \mid a_i$ property for n elements is based on the divisibility properties of numbers and their divisors, and Euler’s theorem [GKP89].

We first prove several lemmas.

Let p be a positive divisor of an odd positive integer n . Define

$$[n]_p = \left\{ p, 2p, \dots, \left(\frac{n}{p} - 1\right)p \right\} \quad [n]_p^* = \left\{ ip \in [n]_p : \left(i, \frac{n}{p}\right) = 1 \right\}$$

where (a, b) is the greatest common divisor of a and b .

Lemma 1. $\dots \left\{ [n]_p^* : p \mid n \text{ and } 1 \leq p < n \right\} \dots [n - 1]$

First we show that

$$[n - 1] = \bigcup_{p \mid n} [n]_p^*$$

Let $m \in [n - 1]$ and define $d = (m, n)$. Then $\left(\frac{m}{d}, \frac{n}{d}\right) = 1$, so $m = \frac{m}{d} \cdot d \in [n]_d^*$.

It suffices to show that $[n]_p^* \cap [n]_q^* = \emptyset$, for two different positive divisors p and q of n . Suppose it is not true and let $m \in [n]_p^* \cap [n]_q^*$. Then $m = ip = jq$, where $(i, n/p) = 1$ and $(j, n/q) = 1$. Denote $d = (p, q)$. Clearly, there are positive integers p_1 and q_1 such that $p = dp_1$, $q = dq_1$, and $(p_1, q_1) = 1$. As $ip = jq$, $idp_1 = jdq_1$, so $ip_1 = jq_1$. Since $(p_1, q_1) = 1$, $q_1 \mid i$. On the other hand, $q \mid n$, so $n = qn_1$ for some positive integer n_1 . Consequently,

$$\frac{n}{p} = \frac{qn_1}{dp_1} = \frac{dq_1n_1}{dp_1} = q_1 \frac{n_1}{p_1}$$

(as n/p is an integer and $(p_1, q_1) = 1$, n_1/p_1 is itself an integer). Hence $q_1 \mid n/p$ and $q_1 \mid i$, so $q_1 = 1$ because $(i, n/p) = 1$. For a similar reason $p_1 = 1$ and consequently $p = d = q$, a contradiction. ■

We define a directed graph G_n on $[n - 1]$. A pair (i, j) is an edge in G_n if $j \equiv 2i \pmod{n}$, for $i, j \in [n - 1]$. Notice that G_n is well-defined. Indeed, suppose that $2i \equiv 0 \pmod{n}$ for some $i \in [n - 1]$. As n is odd, it implies $i \equiv 0 \pmod{n}$, a contradiction. Clearly, for each $i \in [n - 1]$, there is a unique $j \in [n - 1]$ such that (i, j) is an edge in G_n . Hence the outdegrees of vertices in G_n are all equal to 1.

As n is odd, by Euler's theorem [GKP89] we have $2^{\varphi(n)} \equiv 1 \pmod{n}$, where $\varphi(n)$ is Euler's function [GKP89]. Let $j \in [n - 1]$ and define $i \equiv 2^{\varphi(n)-1} \cdot j \pmod{n}$. Then

$$2i = 2 \cdot \left(2^{\varphi(n)-1} \cdot j\right) = 2^{\varphi(n)} \cdot j \equiv j \pmod{n}$$

Hence the indegree of each vertex in G_n is at least 1. Since the sum of the indegrees of all vertices in a directed graph is equal to the sum of the outdegrees, the indegree of each vertex is 1. Therefore the components of G_n are cycles.

Lemma 2. $\dots p \mid n, 1 \leq p < n, \dots t_p \dots t > 1 \dots [n]_p^* \dots t_p$

Let $s \in [n]_p^*$. Then $s = ip$, $1 \leq i < \frac{n}{p}$ and $(i, n/p) = 1$.

If $1 \leq i \leq \frac{n}{2p}$, then $2s = 2ip < n$ (because n is odd), so $2s \in [n]_p$. If $\frac{n}{2p} < i < \frac{n}{p}$ then $2s = 2ip \equiv 2ip - n = (2i - n/p)p \pmod{n}$ and $1 \leq 2i - \frac{n}{p} < 2\frac{n}{p} - \frac{n}{p} = \frac{n}{p}$. Hence $2s - n \in [n]_p$. We have shown that $2s \pmod{n}$ belongs to $[n]_p$. Let $d = (2i, n/p)$, when $1 \leq i \leq \frac{n}{2p}$ and $d = (2i - n/p, n/p)$, when $\frac{n}{2p} < i < \frac{n}{p}$. Since n/p is odd and $(i, n/p) = 1$ we get $d = 1$.

We proved that the cycle in G_n containing s has all vertices in $[n]_p^*$. The length of this cycle is the smallest integer t such that $s \equiv 2^t \cdot s \pmod{n}$. This is equivalent to $ip \equiv 2^t \cdot ip \pmod{n}$, for some $1 \leq i < n/p$, which in turn is equivalent to $(2^t - 1)i \equiv 0 \pmod{n/p}$. Since $(i, n/p) = 1$, the last condition is equivalent to $2^t \equiv 1 \pmod{n/p}$. Hence the length of the cycle containing s , and consequently the length of any cycle in the subgraph of G_n induced by the vertices of $[n]_p^*$, is t_p . ■

Lemma 3. $G_n \dots \frac{5n}{\log_2 n}$

Let $p \mid n$, $1 \leq p < n$. Observe first that, as $2^{t_p} \equiv 1 \pmod{n/p}$ and $t_p > 1$, $2^{t_p} \geq n/p + 1$. Hence $t_p \geq \log_2(n/p)$. Thus the number of cycles in the subgraph of G_n induced by the set of vertices $[n]_p^*$ is at most $\frac{1}{\log_2(n/p)} \cdot \varphi(n/p)$. Consequently, the total number of cycles in G_n is at most

$$\begin{aligned} \sum_{\substack{p \mid n \\ 1 \leq p < n}} \frac{1}{\log_2 \frac{n}{p}} \varphi\left(\frac{n}{p}\right) &= \sum_{\substack{p \mid n \\ 1 < p \leq n}} \frac{\varphi(p)}{\log_2 p} = \sum_{\substack{p \mid n \\ 1 < p < n^{\frac{1}{3}}}} \frac{\varphi(p)}{\log_2 p} + \sum_{\substack{p \mid n \\ n^{\frac{1}{3}} \leq p \leq n}} \frac{\varphi(p)}{\log_2 p} \\ &\leq \sum_{\substack{p \mid n \\ 1 < p < n^{\frac{1}{3}}}} n^{\frac{1}{3}} + \sum_{\substack{p \mid n \\ n^{\frac{1}{3}} \leq p \leq n}} \frac{\varphi(p)}{\log_2 n^{\frac{1}{3}}} \leq n^{\frac{2}{3}} + \frac{1}{\frac{1}{3} \log_2 n} \cdot \sum_{\substack{p \mid n \\ 1 < p \leq n}} \varphi(p) \\ &= n^{\frac{2}{3}} + \frac{3}{\log_2 n} \cdot n \leq \frac{5n}{\log_2 n}. \end{aligned}$$

Let \overline{G}_n be a directed graph on $[(n-1)/2]$ such that, for $i, j \in [(n-1)/2]$, (i, j) is an edge if

$$j = \begin{cases} 2i, & \text{if } 2i \leq \frac{n-1}{2} \\ n-2i, & \text{if } 2i > \frac{n-1}{2}. \end{cases}$$

Clearly, the outdegree of every vertex in \overline{G}_n is 1. Let $j \in [(n-1)/2]$. If j is even, i.e. $j = 2i$ for some $i \in [(n-1)/2]$, then (i, j) is an edge in \overline{G}_n . If j is odd then let $i = (n-j)/2$. Since $2i = n-j > (n-1)/2$ and $n-2i = j$, (i, j) is an edge in \overline{G}_n . We have shown that every vertex in \overline{G}_n has outdegree and indegree equal to 1.

Let $s \in [(n-1)/2]$ and denote by \overline{C} the cycle in \overline{G}_n containing s . Let $C = \{c_0, c_1, \dots, c_{t-1}\}$ be a cycle in G_n containing s , $c_j \equiv 2^j s \pmod{n}$. We shall show

by induction on j that $\bar{c}_j \in \bar{C}$, where $\bar{c}_j = \min(c_j, n - c_j)$, $j = 0, 1, \dots, t - 1$. Obviously, $\bar{c}_0 = s \in \bar{C}$. Assume that $\bar{c}_0, \bar{c}_1, \dots, \bar{c}_{j-1} \in \bar{C}$, $j < t$. We shall show that $\bar{c}_j \in \bar{C}$. Let us consider four cases.

Case 1: $c_{j-1} \leq (n - 1)/2$ and $c_j \leq (n - 1)/2$

Then $\bar{c}_{j-1} = c_{j-1}$, $\bar{c}_j = c_j$, and $2\bar{c}_{j-1} = \bar{c}_j \leq (n - 1)/2$, so $(\bar{c}_{j-1}, \bar{c}_j)$ is an edge in \bar{G}_n . Consequently, $\bar{c}_j \in \bar{C}$ (as $\bar{c}_{j-1} \in \bar{C}$).

Case 2: $c_{j-1} \leq (n - 1)/2$ and $c_j > (n - 1)/2$

Then $\bar{c}_{j-1} = c_{j-1}$ and $\bar{c}_j = n - c_j$. Observe that $2\bar{c}_{j-1} = 2c_{j-1} = c_j > (n - 1)/2$ and $n - 2\bar{c}_{j-1} = n - c_j = \bar{c}_j$, so $(\bar{c}_{j-1}, \bar{c}_j)$ is an edge in \bar{G}_n . Consequently, $\bar{c}_j \in \bar{C}$.

Case 3: $c_{j-1} > (n - 1)/2$ and $c_j \leq (n - 1)/2$

Then $\bar{c}_{j-1} = n - c_{j-1}$ and $\bar{c}_j = c_j$. Moreover $2c_{j-1} = n + c_j$. Observe that

$$\begin{aligned} 2\bar{c}_{j-1} &= 2(n - c_{j-1}) = 2n - 2c_{j-1} = 2n - (n + c_j) = n - c_j \\ &\geq n - \frac{n - 1}{2} = \frac{n + 1}{2} \end{aligned}$$

and $n - 2\bar{c}_{j-1} = c_j = \bar{c}_j$, so $(\bar{c}_{j-1}, \bar{c}_j)$ is an edge in \bar{G}_n . Consequently, $\bar{c}_j \in \bar{C}$.

Case 4: $c_{j-1} > (n - 1)/2$ and $c_j > (n - 1)/2$

Then $\bar{c}_{j-1} = n - c_{j-1}$, $\bar{c}_j = n - c_j$, and $2c_{j-1} = n + c_j$. Moreover,

$$2\bar{c}_{j-1} = 2n - 2c_{j-1} = 2n - n - c_j < n - \frac{n - 1}{2} = \frac{n + 1}{2}$$

so $2\bar{c}_{j-1} \leq \frac{n-1}{2}$. Since $2\bar{c}_{j-1} = n - c_j = \bar{c}_j$, we have that $(\bar{c}_{j-1}, \bar{c}_j)$ is an edge in \bar{G}_n , and thus $\bar{c}_j \in \bar{C}$.

We proved that $\{\bar{c}_0, \bar{c}_1, \dots, \bar{c}_{t-1}\} \subseteq \bar{C}$ and that $(\bar{c}_{j-1}, \bar{c}_j)$ is an edge in \bar{G}_n , for $j = 1, 2, \dots, t - 1$. It is easy to show (proceeding as in cases 1 and 3) that $(\bar{c}_{t-1}, \bar{c}_0)$ is an edge in \bar{G}_n . Thus

$$\{\bar{c}_0, \bar{c}_1, \dots, \bar{c}_{t-1}\} = \bar{C}$$

We have proved that the number of cycles in \bar{G}_n is not larger than in G_n . By Lemma 3 we get the next lemma.

Lemma 4. $\dots \dots \dots \bar{G}_n \dots \dots \dots \frac{5n}{\log_2 n}$

Now a sequence with the $\dots \dots \dots$ property can be defined as follows.

We choose from each cycle C in \bar{G}_n $\lfloor \frac{1}{2} \ell_c \rfloor$ pairs of vertices joined by an edge, where ℓ_c is the length of C . This way, letting C range over the cycles of \bar{G}_n , we get at least

$$\sum_C \left\lfloor \frac{1}{2} \ell_c \right\rfloor \geq \sum_C \left(\frac{1}{2} \ell_c - \frac{1}{2} \right) \geq \frac{1}{2} \sum_C \ell_c - \frac{1}{2} \frac{5n}{\log_2 n} = \frac{1}{2} \frac{n - 1}{2} - \frac{1}{2} \frac{5n}{\log_2 n}$$

pairwise disjoint pairs (by Lemma 4). Denote by \mathcal{A} the set of i s in the chosen pairs.

For each such pair (i, j) we create a sequence s_i , which is a concatenation of the following d sequences, where $d = d_i = (i, n)$:

- $0, i, 2i, \dots, \left(\frac{n}{d} - 1\right) i, 0, i \pmod n$
- $1, i + 1, 2i + 1, \dots, \left(\frac{n}{d} - 1\right) i + 1, 1, i + 1 \pmod n$
- $2, i + 2, 2i + 2, \dots, \left(\frac{n}{d} - 1\right) i + 2, 2, i + 2 \pmod n$
- \dots
- $d - 1, i + d - 1, 2i + d - 1, \dots, \left(\frac{n}{d} - 1\right) i + d - 1, d - 1, i + d - 1 \pmod n.$

For $\alpha, \beta \in \{0, 1, \dots, n - 1\}$, define the distance

$$\text{dist}(\alpha, \beta) = \text{dist}(\beta, \alpha) = \min(|\beta - \alpha|, n - |\beta - \alpha|).$$

Observe that every pair in $\{0, 1, \dots, n - 1\}$ for which the distance is i occurs in s_i as consecutive objects; and each pair for which the distance is j occurs in s_i within a distance of 2. Moreover, the length of s_i is $n + 2d$.

The number of vertices in \overline{G}_n which are not in any of the chosen pairs (i, j) is at most

$$\frac{n - 1}{2} - \left(\frac{n - 1}{2} - \frac{5n}{\log_2 n}\right) = \frac{5n}{\log_2 n}.$$

For each such vertex ℓ we create a sequence s_ℓ given by the same definition as s_i (replacing i with ℓ). Denote the set of these vertices ℓ by \mathcal{B} .

It is easily seen that a concatenation s of all sequences s_i and s_ℓ has the 2-radius property. The length of s is

$$\begin{aligned} \sum_{i \in \mathcal{A} \cup \mathcal{B}} (n + 2d_i) &= \sum_{i \in \mathcal{A}} (n + 2d_i) + \sum_{i \in \mathcal{B}} (n + 2d_i) \leq |\mathcal{A}| \cdot n + 2 \sum_{i \in \mathcal{A}} d_i + |\mathcal{B}| \cdot 2n \\ &\leq \frac{1}{2} \frac{n - 1}{2} n + \frac{5n}{\log_2 n} \cdot 2n + 2 \sum_{i=1}^{n-1} d_i = \frac{1}{2} \binom{n}{2} + \frac{10n^2}{\log_2 n} + 2 \sum_{i=1}^{n-1} d_i. \end{aligned}$$

Let us estimate the sum $\sum_{i=1}^{n-1} d_i$.

$$\sum_{i=1}^{n-1} d_i = \sum_{\substack{p|n \\ 1 \leq p < n}} p \cdot |\{i : (i, n) = p\}| = \sum_{\substack{p|n \\ 1 \leq p < n}} p \cdot \varphi\left(\frac{n}{p}\right) \leq \sum_{\substack{p|n \\ 1 \leq p < n}} p \cdot \frac{n}{p} = \sum_{\substack{p|n \\ 1 \leq p < n}} n.$$

Let $n = p_1 p_2 \dots p_r$ be a factorization of n into (not necessarily distinct) primes. The number of divisors of n is then not larger than 2^r . As n is odd, $n = p_1 p_2 \dots p_r \geq 3^r$, so $2^r \leq 2^{\log_3 n} = 2^{\log_2 n \cdot \log_3 2} = n^{\log_3 2} \leq n^{0.64}$. Consequently $\sum_{i=1}^{n-1} d_i \leq n^{1.64}$ and the length of s is at most

$$\frac{1}{2} \binom{n}{2} + \frac{10n^2}{\log_2 n} + 2n^{1.64} = \frac{1}{2} \binom{n}{2} (1 + o(1)).$$

If n is even then we construct the sequence s for $n - 1$ and concatenate it with the sequence $1, 2, n, 3, 4, 5, 6, n, 7, 8, 9, 10, n, \dots$ whose length is not larger than $n - 1 + \lceil \frac{n-1}{4} \rceil$.

4 A General Construction Using a 1-Radius Sequence

Let $M = \lceil \frac{n}{\lfloor (k+1)/2 \rfloor} \rceil$ and $X = \{x_1, x_2, \dots, x_M\}$. Consider a sequence p of length $f_1(M)$ in which each pair of elements from X occurs consecutively. We partition the set $[n]$ into M disjoint subsets of cardinality $\lfloor \frac{k+1}{2} \rfloor$ except for at most one of a smaller cardinality. Denote these subsets by A_1, A_2, \dots, A_M .

Denote by s_i any sequence (permutation) of elements of A_i . Define s to be the sequence obtained from p by replacing every occurrence of x_i by the sequence s_i (for every $i = 1, 2, \dots, M$). Observe that each two elements $r, t \in [n]$ occur within distance at most k in the sequence s . Indeed, let $r \in A_p \subseteq [n]$ and $t \in A_q \subseteq [n]$. The elements $x_p, x_q \in X$ are at least once neighbors in the sequence p . Since $|s_p| \leq \lfloor \frac{k+1}{2} \rfloor$ and $|s_q| \leq \lfloor \frac{k+1}{2} \rfloor$, r and t are in s within distance at most $2 \lfloor \frac{k+1}{2} \rfloor - 1 \leq k$.

We proved that $f_k(n) \leq |s|$. Let us compute $|s|$. If M is odd then

$$\begin{aligned} |s| &\leq f_1(M) \cdot \left\lfloor \frac{k+1}{2} \right\rfloor \leq \left(\binom{M}{2} + 1 \right) \left\lfloor \frac{k+1}{2} \right\rfloor = \left(\frac{1}{2}M^2 - \frac{1}{2}M + 1 \right) \left\lfloor \frac{k+1}{2} \right\rfloor \\ &\leq \left(\frac{1}{2} \cdot \frac{(n + \lfloor (k+1)/2 \rfloor)^2}{\lfloor (k+1)/2 \rfloor^2} - \frac{1}{2} \cdot \frac{n + \lfloor (k+1)/2 \rfloor}{\lfloor (k+1)/2 \rfloor} + 1 \right) \cdot \left\lfloor \frac{k+1}{2} \right\rfloor \\ &= \frac{1}{2} \frac{n^2}{\lfloor (k+1)/2 \rfloor} + \frac{1}{2}n + \left\lfloor \frac{k+1}{2} \right\rfloor. \end{aligned}$$

If M is even, we have

$$\begin{aligned} |s| &\leq \left(\binom{M}{2} + \frac{1}{2}M \right) \left\lfloor \frac{k+1}{2} \right\rfloor = \frac{1}{2} \left[\frac{n}{\lfloor (k+1)/2 \rfloor} \right]^2 \cdot \left\lfloor \frac{k+1}{2} \right\rfloor \\ &\leq \frac{1}{2} \cdot \frac{(n + \lfloor (k+1)/2 \rfloor)^2}{\lfloor (k+1)/2 \rfloor^2} \cdot \left\lfloor \frac{k+1}{2} \right\rfloor = \frac{1}{2} \frac{n^2}{\lfloor (k+1)/2 \rfloor} + n + \frac{1}{2} \left\lfloor \frac{k+1}{2} \right\rfloor. \end{aligned}$$

Consequently, we have the following:

Theorem 4. $f_k(n) \leq \frac{n^2}{2\lfloor (k+1)/2 \rfloor} + n + \frac{1}{2} \lfloor \frac{k+1}{2} \rfloor$.

5 A General Construction Using a 2-Radius Sequence

Let $M = \lceil \frac{n}{\lfloor (k+1)/3 \rfloor} \rceil$ and $X = \{x_1, \dots, x_M\}$. Consider the sequence p of length $f_2(M)$ in which two elements of X occur within distance at most 2. We partition the set $[n]$ into M disjoint subsets of cardinality $\lfloor \frac{k+1}{3} \rfloor$, except for at most one of a smaller cardinality. Call these subsets A_1, A_2, \dots, A_M .

As in the previous construction we denote by s_i any sequence of elements of A_i . The sequence s is defined as in the previous construction.

Observe that each two elements $r, t \in [n]$ occur within distance at most k in the sequence s . Indeed, let $r \in A_p \subseteq [n]$ and $t \in A_q \subseteq [n]$. As x_p and x_q occur

within distance at most 2 in p, r and t are in s within a distance of $3 \lfloor \frac{k+1}{3} \rfloor - 1 \leq k$ of each other.

We have proved that

$$f_k(n) \leq f_2(M) \cdot \left\lfloor \frac{k+1}{3} \right\rfloor = |s|.$$

Let us estimate $|s|$, for a fixed k .

$$|s| = \left\lfloor \frac{k+1}{3} \right\rfloor \cdot \frac{1}{4} \left(\frac{n}{\lfloor (k+1)/3 \rfloor} \right)^2 (1 + \dots (1)) = \frac{1}{4} \frac{n^2}{\lfloor (k+1)/3 \rfloor} (1 + \dots (1)).$$

We have the following theorem.

Theorem 5.

$$f_k(n) \leq \frac{n^2}{4 \lfloor (k+1)/3 \rfloor} (1 + \dots (1))$$

To illustrate the algorithm, consider a set $A = \{a, b, c, d, e, f, g, h, i, j, k, l\}$ of cardinality $n = 12$. Take $k = 5$. To construct a sequence with the 5-radius property, we can use a sequence $\boxed{1} \boxed{2} \boxed{3} \boxed{4} \boxed{5} \boxed{6} \boxed{1} \boxed{2} \boxed{4} \boxed{5} \boxed{3} \boxed{6}$ with the 2-radius property for $M = 6$ objects. Since $k = 5$, we partition A into groups of size $\frac{6}{3} = 2$ each, for example by taking consecutive pairs of letters, such as $A_1 = \{a, b\}$, $A_2 = \{c, d\}, \dots, A_6 = \{k, l\}$. Following the algorithm, we replace each number i with A_i . This results in $\boxed{a, b} \boxed{c, d} \boxed{e, f} \boxed{g, h} \boxed{i, j} \boxed{k, l} \boxed{a, b} \boxed{c, d} \boxed{g, h} \boxed{i, j} \boxed{e, f} \boxed{k, l}$ (i.e., $abcdefghijklabcdghijefkl$), which has the 5-radius property.

The reasoning applied in the two constructions can be easily generalized to prove the following proposition.

Proposition 1. . . . $k \geq m,$

$$f_k(n) \leq f_m \left(\left\lceil \frac{n}{\lfloor (k+1)/(m+1) \rfloor} \right\rceil \right) \cdot \left\lfloor \frac{k+1}{m+1} \right\rfloor$$

Corollary 2.

$$f_1(n) = \frac{1}{2}n^2 (1 + \dots (1)). \quad f_2(n) = \frac{1}{4}n^2 (1 + \dots (1)).$$

.....

$$\frac{1}{2k}n^2 (1 + \dots (1)) \leq f_k(n) \leq \frac{1}{4 \cdot \lfloor (k+1)/3 \rfloor}n^2 (1 + \dots (1)).$$

..... $k > 2$

6 Other Strategies

Above we have focused on a particular strategy based on a FIFO (of size $k + 1$) that removes the oldest element and places the new element at the end of the sequence. While this strategy seems to be natural for streaming and transmission of data, other strategies exist that can be useful for other applications. In the general case, we could allow the removal of any element from the current sequence of $k + 1$ objects, placing the new element in an arbitrary position in this sequence. Such a problem is of interest in caching when costs may depend on what locations in the cache are affected. In a more specific case, the replacement could follow the LIFO strategy, which replaces elements by popping one or more of the most recently inserted objects and then pushing new elements. Such a strategy lends itself to a simple construction of a sequence: divide the set into disjoint groups of k elements (except perhaps for the last group that can be smaller); place a group in the memory and add as the $(k + 1)$ st element, consecutively, all the remaining elements from not yet processed groups; when finished, replace the group with the next group and repeat the process until all the groups have been in the memory. This leads to a sequence of the length similar to the FIFO strategy, but occasionally all the elements in the memory must be replaced.

7 Conclusions

Sequences with the k -radius property are useful in scheduling the fetching of data to process each pair of objects—e.g., to compute two-argument functions—when the memory is limited and the objects are huge. As such they are applicable to streaming and caching images or arrays and they help optimize the transmission time or the number of memory rewrites when the application requires access to every pair of objects. The presented results demonstrate asymptotically optimal constructions for sequences with the k -property and demonstrate that the length of such sequences approximately halves when the memory size doubles. Additional constructions based on Steiner systems and finite geometries are presented in [JL04]. One natural generalization of the problem is to consider shortest sequences where all the triples, or quadruples (etc.) appear within radius k at least once.

Acknowledgement

The first author would like to thank J. Gilkerson for discussions that led to the formulation of the problem, N. Imam for experimental implementations for small-radius sequences, and N. Moore for help with typesetting and proofreading.

References

- [CR99] Colbourn, Ch., J., Rosa, A., Triple Systems, Clarendon Press - Oxford, 1999.
- [GJ02] Gilkerson, J. W., Jaromczyk, J. W., Restoring the order of images in a sequence of MRI slices, Manuscript, 2002.
- [G75] Gosh, S.P., Consecutive storage of relevant records with redundancy, *Communications of the ACM* 18(8):464-471, 1975.
- [GGL95] Graham, R. L., Grötschel, M., Lovász, L., Handbook of Combinatorics, The MIT Press - North Holland, 1995.
- [GKP89] Graham, R. L., Knuth, D. E., Patashnik, O., Concrete Mathematics, Addison-Wesley, 1989.
- [JL04] Jaromczyk, J. W., Lonc, Z., Sequences of radius k . *Technical Report TR 417-04, Computer Science - University of Kentucky*, 2004.
- [LTT81] Lonc Z., Traczyk T. Truszczyński, M., Optimal f -graphs for the family of all k -subsets of an n -set. Data base file organization (Warsaw, 1981), 247-270, Notes Rep. Comput. Sci. Appl. Math., 6, Academic Press, New York, 1983.
- [SChD02] Sen, S., Chatterjee, S., Dumir, N., Towards a theory of cache-efficient algorithms, *Journal of the ACM*, 49(6):828-858, 2002.

New Bounds on Map Labeling with Circular Labels^{*}

Minghui Jiang¹, Sergey Bereg², Zhongping Qin³, and Binhai Zhu¹

¹ Department of Computer Science, Montana State University,
Bozeman, MT 59717-3880, USA
{jiang,bhz}@cs.montana.edu

² Department of Computer Science, University of Texas at Dallas,
Richardson, TX 75083-0688, USA
besp@utd.edu

³ College of Software, Huazhong University of Science and Technology,
Wuhan, China

Abstract. We present new approximation algorithms for the NP-hard problems of labeling points with maximum-size uniform circles and circle pairs (MLUC and MLUCP). Our algorithms build on the important concept of *maximal feasible region* and new algorithmic techniques. We obtain several results: a $(2.98 + \epsilon)$ -approximation for MLUC, improving previous factor $3.0 + \epsilon$; a $(1.491 + \epsilon)$ -approximation for MLUCP, improving previous factor 1.5; and the first non-trivial lower bound 1.0349 for both MLUC and MLUCP, improving previous lower bound $1 + O(1/n)$.

1 Introduction

Map labeling is a geometric optimization problem: Given a set of feature sites in the plane, and a label to be placed near each site, the goal is to label the maximum number of sites with the maximum label size such that no labels overlap. The labels are typically axis-parallel rectangles containing textual information. In this paper, we study map labeling with circular labels. Circular labels are useful for conveying graphic information; their natural relation to the Euclidean metric in planar geometry also makes them interesting for theoretical study. We study two closely related problems:

Instance: Given n point sites $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ in the plane, and a label size $r > 0$.

MLUC: Is there a placement for n uniform open circles of radii r , one circle for each input site $P_i \in \mathcal{P}$, such that each site is on the boundary of its labeling circle and no circles intersect?

MLUCP: Is there a placement for $2n$ uniform open circles of radii r , two circles for each input site $P_i \in \mathcal{P}$, such that each site is on both boundaries of its two labeling circles and no circles intersect?

^{*} This research is partially supported by NSF CARGO grant DMS-0138065.

The MLUC problem was first studied by Doddi et al. [2]; they proposed a 29.86-approximation for maximizing the label size such that all points are labeled. This factor was improved to 19.35 by Strijk and Wolff [10]. Later, Doddi et al. [3] introduced the concept of *maximal feasible region*, and improved the approximation factor to $3.6 + \epsilon$. Recently, Jiang et al. [5] presented a conceptually simple $(3 + \epsilon)$ -approximation; this was achieved by proving a combinatorial lemma on labeling disjoint unit circles with points, and by generalizing the concept of feasible region further to *maximal feasible region*. In this paper, we present a $(2.98 + \epsilon)$ -approximation for this problem; our algorithm builds on the important concept of maximal feasible region, and handles the previous difficult cases with new algorithmic techniques.

The MLUCP problem was first studied by Zhu and Poon [12] as part of the research effort in multi-label map labeling. They proposed a 2-approximation [12] for maximizing the label size. This bound was subsequently improved to 1.96 [8], 1.686 [9], and 1.5 [11]. In this paper, we use the concept of maximal feasible region to design a remarkably simple $(1.5 + \epsilon)$ -approximation for MLUCP, then improve the approximation factor to $1.491 + \epsilon$ with new algorithmic techniques.

For the hardness of approximation, Strijk and Wolff [10] proved that the MLUC decision problem is NP-hard; their proof also implied that it is NP-hard to approximate the MLUC optimization problem (maximizing the label size) within factor $1 + O(1/n)$. For labeling points with maximum-size uniform circle pairs, a 1.37 inapproximability result [8] was claimed (no details were given); it was later found to be actually $1 + O(1/n)$. In this paper, we design a new reduction to obtain the first nontrivial lower bound of 1.0349 for both MLUC and MLUCP. We summarize our results in the following table.

problem	upper bound	lower bound
MLUC	$3 + \epsilon$ [5] $\rightarrow 2.98 + \epsilon$	$1 + O(1/n)$ [10] $\rightarrow 1.0349$
MLUCP	1.5 [11] $\rightarrow 1.491 + \epsilon$	$1 + O(1/n)$ [8] $\rightarrow 1.0349$

2 Maximal Feasible Region and Distance Graph

In this section, we introduce some preliminary concepts. We refer to a circle of unit radius as a unit circle. When we say that a circle has distance d to a point, we mean that the distance from the circle center to the point is d ; similarly for the distance between two circles. When two sites have intersecting candidate circles, we say that the two sites *intersect* each other.

In the MLUC problem, a site can be anywhere on the boundary of its labeling circle—from a different perspective, the circle rotates around a fixed pivot on its boundary that coincides with the site. The position of a circle, given its radius, is determined by its direction: the vector from the site to the circle center. If a circle contains a site in its interior, then it always intersects the circle of the contained site; therefore, a circle at a *feasible position* contains no site in its interior. As a circle rotates continuously from one feasible position to another, the corresponding direction vector sweeps a cone, or, a *feasible cone*.

We consider only *maximal feasible regions*, in the sense that no maximal feasible region is a proper subset of another feasible region (this concept was introduced by Jiang et al. [5]). Because a site is always on the boundary of its circle, which, at a feasible position, contains no site in its interior, each site is the closest site to its circle in a non-intersecting label placement. Maximal feasible regions are naturally related to the Voronoi diagram: the center of a labeling circle is within the Voronoi cell of its site if and only if the circle is at a feasible position.

The MLUCP problem can be considered as a special case of the MLUC problem where each MLUCP site is two coinciding MLUC sites. To avoid overlapping, the two circles of each site must always be labeled in opposite directions, and each site is at the midpoint of its two circle centers. To define maximal feasible region for MLUCP, we first prove the following circle packing property.

Lemma 1.

$$d \leq \sqrt{3} - 1, \quad D(d) = \sqrt{5 - 4 \cos(\frac{\pi}{3} - \phi)} - 1, \quad \phi = \cos^{-1} \frac{5 - (1+d)^2}{4}$$

We refer to Fig. 1. Sites $X, Y,$ and Z are on the boundaries of three circles centered at points $A, B,$ and $C,$ respectively. Y and Z have distance $d \leq \sqrt{3} - 1.$ Without loss of generality, we assume that $XY < XZ.$ We show that XY has the lower bound $D(d).$ Imagine that a spring connects X and $Y,$ and that a rigid rod of length d connects Y and $Z.$ The three circles may not intersect but can move freely otherwise. When the spring shrinks and XY reaches minimum, the points $A, X,$ and Y are collinear; the points $C, Z,$ and Y are collinear; and the three circles are tightly packed in an equilateral triangular formation. \square

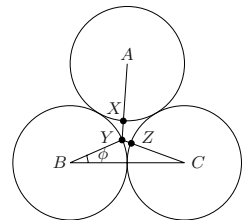


Fig. 1.

The function $D(d)$ is monotonically decreasing in $[0, \sqrt{3} - 1];$ it reaches maximum $\sqrt{3} - 1$ at $d = 0,$ and minimum 0 at $d = \sqrt{3} - 1.$ For completeness, we define $D(d) = 0$ for $d > \sqrt{3} - 1.$ A calculation shows that the equation $D(d) = d$ has solution $d_0 \simeq 0.24.$ For any distance $d < d_0,$ the decreasing property of function $D(d)$ guarantees that $D(d) > d.$ This implies the following corollary:

Corollary 1.

$$d \leq d_0$$

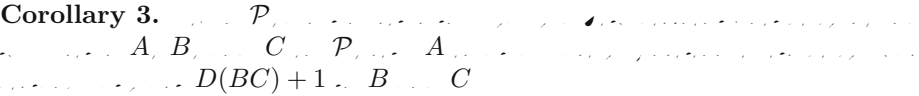
The following corollary defines feasible position for MLUCP.

Corollary 2.

$$d \leq \sqrt{3}$$

In the MLUC problem, maximal feasible regions based on the classic definition [5] exclude obviously infeasible label positions where circles contain other sites in their interiors. The following corollary, immediate from Lemma 1, leads

to more restricted maximal feasible regions that exclude a broader range of infeasible positions.

Corollary 3. 

We revise our definitions of feasible position and maximal feasible region for MLUC based on Corollary 3. We use the term *feasible position* to distinguish this new concept with the *feasible position*.

Given a set of sites \mathcal{P} and a threshold distance d , we define $G_{\mathcal{P}}(d)$ to be the graph where each site is represented by a node, and where an edge connects two nodes if and only if their corresponding sites have distance at most d . We use the abbreviated notation $G(d)$ when the set of sites is clear from context. For threshold distances strictly less than one, the distance graphs have special combinatorial properties if \mathcal{P} can be labeled with disjoint unit circles. An important property on labeling disjoint unit circles with points was discovered by Jiang et al. [5] (Lemma 1); we rephrase this property here using the concept of distance graph:

Lemma 2. 

For the proof of this property, we observe that every simple path in the distance graph $G(1 - \delta)$ has size bounded by $O(1/\delta)$ (an extreme case happens when the path is straight and has unit edge lengths, and each circle is tangent to two consecutive nodes in the path); the $O(1/\delta^2)$ bound follows immediately by an area argument [5]. A tighter bound of $O(1/\delta)$ for the connected component size was recently proved by Bereg [1].

3 A $(2.98 + \epsilon)$ -Approximation for MLUC

In this section, we present a $(2.98 + \epsilon)$ -approximation for MLUC. For completeness, we first sketch a conceptually simple $(3 + \epsilon)$ -approximation [5] that captures some of the underlying ideas: If a set of sites can be labeled with disjoint unit circles, then Lemma 2 implies that, given a small constant δ , where $0 < \delta < 1$, the distance graph $G(1 - \delta)$ has connected components with maximum size bounded by a constant $O(1/\delta^2)$. We use a brute-force search to find a non-intersecting label placement in each component such that each site is labeled in a feasible position. Sites from different components may still have overlapping labels; in the worst case, two sites from different components are labeled with two circles that coincide and are tangent to both sites (the circles are in feasible positions so they contain no sites in their interiors). Because the two sites have distance at least $1 - \delta$ (by the definition of distance graph), the overlapping can be avoided by shrinking all circles to radii $\frac{1}{3 + \epsilon}$, where $\epsilon = O(\delta)$. A binary search on the label size finds a $(3 + \epsilon)$ -approximation.

With the classic concept of maximal feasible region, it is difficult to improve the approximation factor $3 + \epsilon$ further. We refer to Fig. 2. The unit circle centered at O has 6 sites $A, B, C, D, E,$ and F on its boundary in a regular hexagonal formation. A' is another site outside the circle and is very close to A . In the factor- $(3 + \epsilon)$ algorithm [5], these 7 sites are grouped into 6 different components and are labeled independently in their maximal feasible regions. A and A' are in the same component and have to be labeled along vectors \overrightarrow{AO} and $\overrightarrow{OA'}$ (this is the only choice in their maximal feasible regions). The vectors $\overrightarrow{BO}, \overrightarrow{CO}, \overrightarrow{DO}, \overrightarrow{EO},$ and \overrightarrow{FO} are feasible positions of the sites $B, C, D, E,$ and $F,$ respectively; in the worst case they may be labeled along these vectors. When the circles shrink to radii $\frac{1}{3}$, we still obtain a non-intersecting label placement, but there is no room for improvements. If we can label some sites, say, B and $F,$ along directions \overrightarrow{OB} and \overrightarrow{OF} instead, then we will have more space for other sites ($C, D,$ and E) to maneuver. This is exactly the idea behind our new concept of revised maximal feasible region: B and F are close to some closely-clustered sites (A and A'), so their feasible regions should be more restricted.

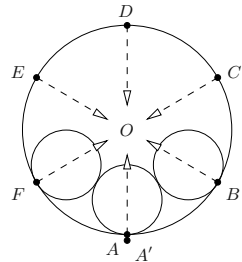


Fig. 2.

Our improved algorithm achieves approximation factor $c + \epsilon$, where $c = 2.98$ and ϵ is a tunable constant that can be arbitrarily small. Let R^* be the optimum label size. Our algorithm finds a non-intersecting label placement with label size at least $\frac{R^*}{c + \epsilon}$ by a binary search with a decision procedure. The decision procedure, given a tentative label size r , either decides that r exceeds $R = R^*(1 - \delta)$ and aborts, or finds a non-intersecting label placement with label size $\frac{r}{c}$. The binary search has range $[R^-, R^+]$ and interval $R^-\delta$, where $R^- < R^* < R^+$ and $\delta = \frac{\epsilon}{2(c + \epsilon)}$. When the binary search converges, we have $\frac{r}{c} \geq \frac{R^*(1 - \delta) - R^-\delta}{c} \geq \frac{R^*(1 - 2\delta)}{c} = \frac{R^*}{c + \epsilon}$.

We first show how the search range is determined. We choose R^+ , the upper bound of the binary search range, to be $(2 + \sqrt{3})D_3$; it is known [2] that $R^* \leq (2 + \sqrt{3})D_3$, where D_3 is the minimum 3-diameter of \mathcal{P} . We choose the lower bound R^- to be $\frac{1}{4}D_3$. We prove $R^* \geq R^-$ by construction: (1) For two sites with distance at least D_3 , their size- $\frac{1}{4}D_3$ circles do not intersect regardless of their orientations. (2) For two sites with distance less than D_3 , their circles cannot intersect circles of other sites, since the distance from other sites to them is at least D_3 (by definition); moreover, their circles do not intersect each other when labeled in opposite directions along the line through the two sites.

We now describe the decision procedure.

1. Compute $G(r)$ and group the sites into components corresponding to the connected components in $G(r)$. If the size of the largest component exceeds the bounding constant $f(\delta)$, abort.
2. With label size r , first compute the revised maximal feasible regions for all sites, then label each component independently. Within each component,

label the sites by brute force, ignoring possible intersections between circles from different components:

- (a) If a site has distance less than d_0r to its two nearest neighbors, abort. Partition the revised maximal feasible regions of each site into cones of maximum angle θ_δ , where $\theta_\delta = \frac{d_0^2}{16}\delta$. If a site P has distance less than d_0r to its nearest neighbor Q , and if a cone C of P contains an internal (non-boundary) vector \mathbf{v} in opposite direction to a boundary vector of a cone of Q , divide cone C into two smaller cones using vector \mathbf{v} .
 - (b) Limit possible label positions of each site to the boundary vectors of its cones. Enumerate all possible combinations to find a non-intersecting label placement. If no such label placement can be found, abort.
3. Make adjustment to the label placement to eliminate intersections between circles from different components:
- (a) Compute distance graph $G(d_cr)$, where $d_c = \sqrt{4 - (\frac{c^2-3}{c})^2} \simeq 0.326$, and group the sites into \dots corresponding to the connected components in $G(d_cr)$. (We call them \dots to avoid confusion with the \dots defined in step 1.) If a cluster contains only one site, we call it a \dots ; otherwise, we call it a \dots .
 - (b) Shrink each circle to size $\frac{r}{c}$ along its direction vector. For each single-site cluster, rotate its circle clockwise for an angle $\alpha = \cos^{-1} \frac{c^2-3}{2c} \simeq 9.37^\circ$. If the resulting label placement has intersection, abort.

Theorem 1. $r \leq R$, \dots

The correctness of step 1 of the decision procedure is immediate from Corollary 2. It remains to prove that, if $r \leq R$, then step 2 always finds a non-intersecting label placement within each component, and step 3 always transforms the output of step 2 into a non-intersecting label placement for all sites.

Lemma 3. \dots

Let P and Q be the two sites. The two circles labeling P and Q are centered at points A and B before shrinking, and at C and D after shrinking. Without loss of generality, we assume that the two circles at A and B touch each other. We have $PQ = d$, $PA = QB = 1$, and $AB = 2$.

Let $\gamma = \max\{\angle PAB, \angle QBA\}$. When the midpoint of AB coincides with the midpoint of PQ , γ reaches minimum $2 \sin^{-1} \frac{d}{4}$. The distance between points C and D along the AB direction is at least $2 - \delta - \delta \cos \gamma$. The two small circles at C and D are separated by a gap of at least $g = (2 - \delta - \delta \cos \gamma) - 2(1 - \delta) = \delta(1 - \cos \gamma)$; each small circle can rotate for an angle $\frac{g}{2}$ without closing the gap. Therefore, we have $\theta = \frac{g}{2} = \frac{\delta}{2}(1 - \cos(2 \sin^{-1} \frac{d}{4})) = \frac{d^2}{16}\delta$. □

Lemma 4. $r \leq R$.

We prove by construction. Given any non-intersecting label placement \mathcal{C} with label size R^* , we construct a non-intersecting label placement \mathcal{C}' with label size R . Since $R < R^*$, each size- R^* circle in \mathcal{C} is labeled in one of the cones that comprise the revised maximal feasible regions at label size R . To obtain \mathcal{C}' , we shrink each circle in \mathcal{C} to size R , and relabel it along one of the two boundary vectors of the cone that contains it.

If a site's nearest neighbor is at least d_0r away, then we choose either vector arbitrarily. The angle between the circle's original continuous direction and its discrete direction is bounded by the maximum cone angle. Lemma 3 implies that, with a rotation angle at most $\theta_\delta = \frac{d_0^2}{16}\delta$, there is no intersection. If a site has distance less than d_0r to its nearest neighbor, then according to Corollary 1 all other sites are d_0r away from the two sites. To label the two sites, there are four possible pairs of boundary vectors from their two cones. If the two cones have a pair of opposing boundary vectors, we label the two sites along the opposing vectors; otherwise, the two cones must have no opposing vectors at all (not even opposing vectors) because of the dividing vectors in step 2a, and we choose the pair of boundary vectors with the largest spreading angle. \square

We next prove the correctness of step 3, that is, if $r \leq R$, then step 3 always transforms the output of step 2 into a non-intersecting label placement for all sites. In particular, we need to show that the non-intersection within each component is maintained, and that the possible intersections between different components are eliminated.

Lemma 5. $d > d_c$, $\theta = \cos^{-1} \frac{d}{4} - \cos^{-1} \frac{cd}{4}$.

This is a stronger form of Lemma 3. Let P and Q be the two sites. The two circles labeling P and Q are centered at points A and B before shrinking, and at C and D after shrinking. Without loss of generality, we assume that the two circles touch each other before shrinking. We have $PQ = d$, $PA = QB = 1$, $PC = QD = \frac{1}{c}$, $AB = 2$, and $CD = \frac{2}{c}$.

We refer to Fig. 3. It is easy to check that the lower configuration gives the minimum $\theta_1 = \cos^{-1} \frac{d}{4} - \cos^{-1} \frac{cd}{4}$ when the two circles rotate in the same direction. On the other hand, if the two circles rotate in opposite directions, the upper configuration gives the minimum $\theta_2 = \sin^{-1}(1 - \frac{d}{2}) - \sin^{-1}(1 - \frac{cd}{4})$. A calculation shows that $\theta_1 \leq \theta_2$. \square

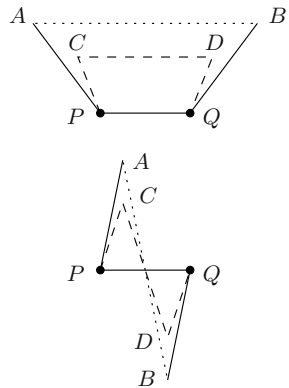


Fig. 3.

Lemma 6. $r \leq R$, $c < 3$, $d_c \geq \frac{c^2 - 3}{4}$, $\alpha \leq \cos^{-1} \frac{c^2 - 3}{2c}$.

Given a site P , if it is from a multi-site cluster, then its circle does not rotate after shrinking. If P is from a single-site cluster, then its nearest neighbor in the same component is at least d_c away. The circle of P shrinks to size $\frac{r}{c}$ and rotate for an angle $\alpha = \cos^{-1} \frac{c^2 - 3}{2c}$. From Lemma 5, this rotation does not cause intersection because d_c satisfies the following equation:

$$\cos^{-1} \frac{c^2 - 3}{2c} = \cos^{-1} \frac{d_c}{4} - \cos^{-1} \frac{cd_c}{4}. \tag{1}$$

□

Lemma 7. $r \leq R$, $c < 3$, $d_c \geq \frac{c^2 - 3}{4}$, $\alpha \leq \cos^{-1} \frac{c^2 - 3}{2c}$.

We refer to Fig. 4. Sites P and Q are from different components. $PQ = e \geq r$; $OP = OQ = O'P = O'Q = r$. An extreme case happens when $PQ = r$ and when P and Q are labeled along \overrightarrow{PO} and \overrightarrow{QO} before the rotation. Even in this extreme case, the two circles do not intersect after shrinking to size $\frac{r}{3}$. If both circles rotate in the same direction for an angle α , a gap will appear between them, which allows both circles to grow a little larger, from size $\frac{r}{3}$ to size $\frac{r}{c}$, without causing intersection (this is exactly the motivation behind our α -rotation). A calculation confirms this.

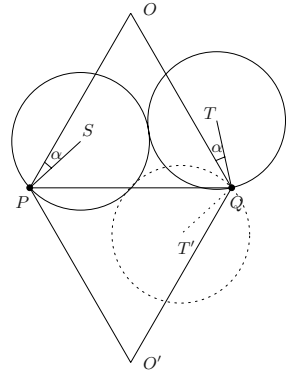


Fig. 4.

Another extreme case happens when Q is labeled along $\overrightarrow{QO'}$ instead. By symmetry, we only need to show that the circle centered at S do not include the midpoint of PQ . This leads to the constraint $\angle SPQ \geq \gamma = \cos^{-1} \frac{ce}{4r}$. A calculation shows that $\alpha + \gamma \leq \angle OPQ = \cos^{-1} \frac{e}{2r}$. □

To satisfy Lemma 7, we need a constant rotation angle $\alpha \geq \cos^{-1} \frac{c^2 - 3}{2c}$ for any constant factor $c < 3$. On the other hand, the maximum rotation angle at each site is determined by the distance d from the site to its nearest neighbor. According to Lemma 5, we must have $\alpha \leq \cos^{-1} \frac{d}{4} - \cos^{-1} \frac{cd}{4}$ (note that $\lim_{d \rightarrow 0} \alpha = 0$). As a compromise of these two constraints, we choose the threshold distant d_c to be the solution of Equation (1), and handle the difficult case of multi-site clusters differently from single-site clusters.

Lemma 8. $r \leq R$, $c < 3$, $d_c \geq \frac{c^2 - 3}{4}$, $\alpha \leq \cos^{-1} \frac{c^2 - 3}{2c}$.

We refer to Fig. 5. Sites P and Q are from two different components. Q is from a multi-site cluster. $PQ = e \geq r$, $OP = OQ = O'P = r$, and $O'Q = (D(d_c) + 1)r$. In the worse case, P is from a single-site cluster, Q 's circle (centered at T) is labeled along \overline{QO} , and P 's circle rotates from $\overline{PO'}$ for an angle α toward Q (its center moves from S' to S). We show that P and Q do not interfere, that is, $ST \geq \frac{2r}{c}$. To simplify the analysis, we prove a stronger claim: let $D_x(X, Y)$ denote the distance along the x -coordinate between points X and Y ; we show that $D_x(S, T) \geq \frac{2r}{c}$.

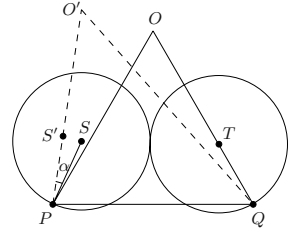


Fig. 5.

Let $\beta = \angle O'PQ$. The segment $\overline{S'S}$ is horizontal (parallel to \overline{PQ}) when $\beta = \frac{\pi}{2} + \frac{\alpha}{2}$. When $\beta \geq \frac{\pi}{2} + \frac{\alpha}{2}$, we have $D_x(P, S) \leq \frac{r}{c} \sin \frac{\alpha}{2}$. Since $D_x(P, T) = e - \frac{e}{2c} \geq r - \frac{r}{2c}$, it follows that

$$D_x(S, T) = D_x(P, T) - D_x(P, S) \geq r - \frac{r}{2c} - \frac{r}{c} \sin \frac{\alpha}{2} \geq \frac{2r}{c}.$$

The last inequality can be checked by calculation.

We next consider the case when $\beta < \frac{\pi}{2} + \frac{\alpha}{2}$. $D_x(S, T)$ is a function of e . As e increases, β becomes smaller, and segment $\overline{S'S}$ tilts farther away from the horizontal position; this implies that $\frac{d}{de} D_x(S', S) \leq 0$. A calculation shows that $\frac{d}{de} D_x(P, S') \leq \frac{3}{2c}$ and that $\frac{d}{de} D_x(P, T) = 1 - \frac{1}{2c}$. Therefore, we have

$$\frac{d}{de} D_x(S, T) = \frac{d}{de} D_x(P, T) - \frac{d}{de} D_x(P, S') - \frac{d}{de} D_x(S', S) \geq 1 - \frac{1}{2c} - \frac{3}{2c} > 0.$$

This inequality implies that the worse case happens when $e = r$. In this worse case, we have $D_x(S, T) = r - \frac{r}{2c} - \frac{r}{c} \cos(\beta - \alpha)$, where $\beta = \cos^{-1} \frac{2 - (D(d_c) + 1)^2}{2}$. Our approximation factor $c = 2.98$ is the smallest c that satisfies the following equation. Note that α and d_c depend on c according to Equation (1).

$$r - \frac{r}{2c} - \frac{r}{c} \cos \left(\cos^{-1} \frac{2 - (D(d_c) + 1)^2}{2} - \alpha \right) = \frac{2r}{c} \tag{2}$$

□

The proof of Lemma 8 also explains why the revised maximal feasible region needs to be introduced in place of the classic maximal feasible region. It is exactly the difference between the two regions, the cone $\angle O'PO$ in Fig. 5, that allows us to handle the difficult case of multi-site clusters.

Theorem 2.

Let $\epsilon \in (0, 1]$. For any $n \geq 2.98 + \epsilon$, there exists a $(2.98 + \epsilon)$ -approximation algorithm for MLUCP with running time $O(n \log n + n(1/\epsilon)^{O(1/\epsilon^2)})$.

4 A $(1.491 + \epsilon)$ -Approximation for MLUCP

In this section, we present approximation algorithms for MLUCP. We first show a very simple algorithm that achieves a 1.5-approximation. Let R^* be the optimum

label size. Our algorithm finds a non-intersecting label placement with label size at least $\frac{R^*}{1.5+\epsilon}$ by a binary search with a decision procedure. The decision procedure, given tentative label size r , either decides that r exceeds R^* and aborts, or finds a non-intersecting label placement with label size $\frac{2}{3}r$:

Compute maximal feasible regions for all sites. If any two sites have distance less than $2r$, or if any site has no feasible position, abort; otherwise, label each site with a circle pair of size $\frac{2}{3}r$ in an arbitrary feasible position.

Theorem 3. $r \leq R^*$, then there exists a non-intersecting label placement with label size $\frac{2}{3}r$.

We refer to Fig. 6(a). The solid and dashed circle pairs have size r , and each labels a site. The large dotted circle has size $2r$ and is centered at the site labeled with the solid circle pair; the two small dotted circles have size $\sqrt{3}r$ and are concentric with the two solid circles. All neighbors of the site must be outside the union of the three dotted circles; therefore, any two sites must have distance at least $2r$.

To show that the size- $\frac{2}{3}r$ labels do not intersect as long as all sites are labeled in feasible positions, we refer to Fig. 6(b) for the extreme case. Sites P and Q has distance $2r$; $S'P = T'Q = r$, $SP = TQ = \frac{2}{3}r$, and $S'Q = T'P = \sqrt{3}r$ (Corollary 2). A calculation shows that $ST = \frac{4}{3}r$; the two size- $\frac{2}{3}r$ circles centered at S and T do not intersect. \square

Our factor- $(1.491 + \epsilon)$ algorithm for MLUCP is almost identical to our factor- $(1.5 + \epsilon)$ algorithm except that the more sophisticated rotation technique is used, as in our factor- $(2.98 + \epsilon)$ algorithm for MLUC. We refer to Fig. 7 and omit the details.

Theorem 4. If $r \leq R^*$, then there exists a non-intersecting label placement with label size $O(n \log n + n \log \frac{1}{\epsilon})$.

5 Lower Bound for Map Labeling with Circular Labels

In this section, we show a new reduction to prove the NP-hardness of MLUC and MLUCP, and obtain the first nontrivial lower bound of 1.0349.

Theorem 5. $R^* \geq 1.0349$.

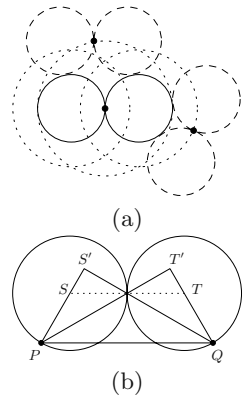


Fig. 6.

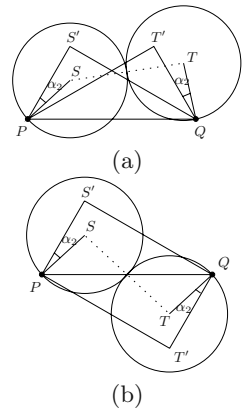


Fig. 7.

We reduce the NP-hard Planar-3SAT problem [7] to MLUCP. The MLUCP instances of the reduction have unit label size. The variable gadget is a four-site cluster as shown in Fig. 8(a). The four sites in the cluster are placed in a square formation with side length $d = 1 + \sqrt{3}$. There are only two ways to label the cluster with unit circle pairs, thereby encoding the boolean value. The clause gadget is shown in Fig. 8(b)(c). Each clause has a circle C centered at a site S , with three sites O, P , and Q on the boundary of C in an equilateral triangular formation. We say that a variable site X is labeled in critical position if site S is collinear with the two centers of the circle-pair labeling X ; the labeling position perpendicular to a critical position is called a perpendicular position. We connect the variable gadgets to the clause gadget in a way such that each variable site on the clause circle have only two labeling choices, either in critical position or in perpendicular position. To complete the description for the clause gadget, the following lemma restricts the radius of the clause circle.

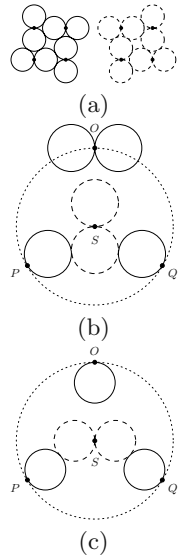


Fig. 8.

Lemma 9.

If the radius of the clause circle is $\frac{3 + \sqrt{13}}{2} \approx 3.303$ and $\frac{2 + \sqrt{3} + \sqrt{15}}{2} \approx 3.802$, then a clause is satisfiable if and only if at least one of its three variable sites is labeled in perpendicular position.

Lemma 9 implies that, with the radius of clause circle set to $\frac{3 + \sqrt{13}}{2}$, a Planar-3SAT clause is satisfiable if and only if there exists a non-intersecting label placement for the clause such that at least one of its three variable sites is labeled in perpendicular position. As in typical reductions from Planar-3SAT [4, 6], we model each variable as an abstract segment—geometrically, a row of four-site clusters. If a variable appears in a clause, we draw a leg from the variable to the clause, as shown in Fig. 9(a). A technical detail in our construction is that the three variable sites on the clause circle must be placed in an equilateral triangular formation and each must be labeled in either critical or perpendicular position; this requirement can be satisfied by twisting each leg, that is, shifting each four-site cluster in the leg by a very small constant amount (the complexity of our construction is inversely related to this constant), such that it eventually reaches the clause at the required position. The layout of a clause and its three variables is shown in Fig. 9(b). Given n clauses in an input Planar-3SAT instance, we need $O(n)$ legs for each variable and $O(n)$ sites to model each variable leg due to the planarity of the input clauses. We also need $O(n^2)$ sites to model the variables as abstract segments. In total, we need $O(n^2)$ sites to encode the n input clauses—consequently, the reduction takes polynomial time.

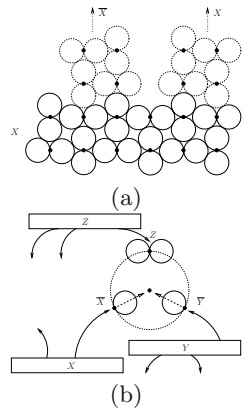


Fig. 9.

To obtain the 1.0349 lower bound, we observe that our reduction still works even if we shrink all the unit circle pairs by a very small amount. After shrinking to size $r = 1/1.0349$, the gaps between the smaller labels allow each variable site's label to rotate for a maximum angle of $\theta = 31.6^\circ$. Since $\theta < 45^\circ$, the correctness of variable encoding is still ensured. To examine the effect of smaller labels on a clause gadget, we note that each variable site on the clause circle is now labeled (that is, at most angle θ away from) either critical or perpendicular position. A case analysis shows that, a Planar-3SAT clause is satisfiable if and only if there exists a non-intersecting label placement for the clause such that at least one of its three variable sites is labeled perpendicular position.

Finally, by replacing every site in the MLUCP problem with a pair of coinciding (or sufficiently close) sites in the MLUC problem, we obtain a reduction from Planar-3SAT to MLUC.

Theorem 6. *Planar 3SAT is reducible to MLUCP with a reduction ratio of 1.0349.*

References

1. Sergey Bereg. Private communication, 2004.
2. Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M. E. Moret, and Binhai Zhu. Map labeling and its generalizations. In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pages 148–157, 1997.
3. Srinivas Doddi, Madhav V. Marathe, and Bernard M. E. Moret. Point set labeling with specified positions. In *Proc. 16th Annual ACM Symposium on Computational Geometry (SoCG'00)*, pages 182–190, 2000.
4. Michael Formann and Frank Wagner. A packing problem with application to lettering of maps. In *Proc. 7th Annual ACM Symposium on Computational Geometry (SoCG'91)*, pages 281–288, 1991.
5. Minghui Jiang, Jianbo Qian, Zhongping Qin, Binhai Zhu, and Robert Cimikowski. A simple factor-3 approximation for labeling points with circles. *Information Processing Letters*, 87(2):101–105, 2003.
6. Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992.
7. David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.
8. Zhongping Qin, Alexander Wolff, Yinfeng Xu, and Binhai Zhu. New algorithms for two-label point labeling. In *Proc. 8th European Symposium on Algorithms (ESA '00)*, LNCS 1879, pages 368–379, 2000.
9. Michael J. Spriggs and J. Mark Keil. A new bound for map labeling with uniform circle pairs. *Information Processing Letters*, 81(1):47–53, 2002.
10. Tycho Strijk and Alexander Wolff. Labeling points with circles. *International Journal of Computational Geometry & Applications*, 11(2):181–195, 2001.
11. Alexander Wolff, Michael Thon, and Yinfeng Xu. A better lower bound for two-circle point labeling. In *Proc. 11th Annual International Symposium on Algorithms and Computation (ISAAC'00)*, LNCS 1969, pages 422–431, 2000.
12. Binhai Zhu and Chung Keung Poon. Efficient approximation algorithms for multi-label map labeling. In *Proc. 10th Annual International Symposium on Algorithms and Computation (ISAAC'99)*, LNCS 1741, pages 143–152, 1999.

Optimal Buffer Management via Resource Augmentation

Jae-Hoon Kim

Department of Computer Engineering,
Pusan University of Foreign Studies, Pusan 608-738, Korea
jhoon@pufs.ac.kr

Abstract. We study a buffer management problem in network switches supporting QoS(Quality of Service). Each input or output port has a buffer of limited capacity which can store up to B packets. Packets arrive online and a buffer management policy determines which packets are transmitted through the buffer. After admitted into the buffer by the policy, packets are transmitted in FIFO fashion, that is, in the order they arrive. Each packet has a value and the goal of the policy is to maximize the total value of packets transmitted.

The main contribution is to apply resource augmentation analysis to the problem, investigating the optimality of various online policies. The online policy has more resources than the optimal offline policy, that is, additional buffer and a higher transmission rate. For two types of models, nonpreemptive and preemptive, we derive optimal online policies with additional buffer or a higher transmission rate. Also we prove lower bounds of the resources for any optimal online policy.

1 Introduction

Recently, the study of buffer management policies for network switches that support the QoS (Quality of Service) has been extensively carried out. In network switches, the input or output ports are equipped with buffers of limited capacity where packets are stored temporarily before transmitted. When incoming traffic exceeds the buffer capacity, packet loss can be happened. The main tasks of the buffer management policy are to determine which arriving packets are admitted into the buffer and which packets already accepted are dropped from the buffer. To realize the QoS, each packet has a value, representing its guaranteed quality of service. The goal of the buffer management policy is to maximize the total value of packets transmitted.

In this work, we abstract the above problem as follows: At each time, an arbitrary set of packets arrives at a buffer and each packet has a positive weight. The buffer can hold at most B packets. At integral time, a packet in the buffer is transmitted. We consider the FIFO model, in which packets are sent in the same order as they arrive. The buffer management policy should decide whether to accept or reject a packet when it arrives. If dropping packets that have been already accepted is allowed, the policy is said to be preemptive, otherwise it

is said to be nonpreemptive. The throughput or gain of a buffer management policy is the total weight of packets transmitted.

We investigate online settings of the problem in which future packet arrivals are unknown. An online buffer management policy makes a decision with no information about packets arriving ahead. In general, no online algorithm can have a performance as good as an optimal offline algorithm. To compensate the online algorithm for the limit of future information, there have been several work [5, 10, 11, 13] originated from [7] which allow the online algorithm more resources, called *resource augmentation*. For our problem, the buffer may have an extra space or a higher transmission rate. We are interested in online policies achieving optimal competitiveness if it is given additional buffer A or it can transmit s times as many packets as an adversary at each transmission time. In other words, the gain of the online buffer policy is at least as much as that of an optimal offline policy.

For the nonpreemptive policy which cannot discard packets already in the buffer, the research was initiated from the work [1], where only two values 1 and α of packets are given. Subsequently, An Zhu [14] presented an online policy whose competitive ratio matches the known lower bound of $\frac{2\alpha-1}{\alpha}$. Also he established matching upper and lower bounds of $\Theta(\log \alpha)$ when the values of packets are arbitrarily given with the maximum value α . For the preemptive case that packets accepted may be dropped, Kesselman et al. [8] gave a 2-competitive policy for arbitrary values packets. Recently, there is an improvement [9] that beats the competitive ratio of two. For two values packets, Zvi Lotker et al. [12] derived nearly optimal competitive ratio of approximately 1.30. In addition, the study of buffer management for a switch with multiple buffers and one output port has been developed in [2, 3, 4, 6].

After a packet is admitted to a buffer, it may be transmitted within B time steps. Actually, the buffer management problem is fairly related to the scheduling problem where a job has an equal processing time, a positive weight, and an uniform deadline. In deadline scheduling literature, if the weights of jobs are equal to their processing times, there were results such as [10, 11] that allow online scheduling algorithms a faster machine or more machines for optimal competitiveness.

To the best of our knowledge, there are only results of Albers et al. [2] that applied the resource augmentation analysis to the buffer management problem. They studied the case that packets have uniform values and there are multiple buffers. In this paper, we deal with the case that packets have arbitrary values and there is a single buffer. We present optimal online policies for nonpreemptive and preemptive model. For nonpreemptive model, if the values of packets are given just as 1 or $\alpha > 1$, we construct an optimal online policy with additional buffer of size B . Also we show a lower bound $(1 - \frac{1}{\alpha})B$ for the size of additional buffer of any optimal online policy, and we prove that having additional buffer of the lower bound size, any optimal online policy needs to double a transmission rate than the optimal offline policy. So we develop an optimal

online policy with additional buffer of size $(1 - \frac{1}{\alpha})B$ and a transmission rate being the 2-fold of the adversary’s one. For arbitrary values packets, we derive an optimal online policy with a buffer of size $2(\lfloor \log \alpha \rfloor + 1)B$ and a transmission rate of $2(\lfloor \log \alpha \rfloor + 1)$ times the adversary’s one. For preemptive model, we show that a greedy policy is optimal with additional buffer of size $\frac{1}{s-1}B$ if it has a transmission rate being the s -fold ($s \geq 2$) of the adversary’s one.

Throughout the paper, for convenience, we assume that transmissions occur only at integral times, and no two packets arrive at each time and no packet arrive at integral times.

2 Lower Bound

We consider a specific instance \mathcal{I} which the adversary present. In \mathcal{I} , all packets arrive during a short period, i.e., before the next transmission time. First, B packets of weight one arrive. And then B packets of weight α follow. In the next theorems, the adversary uses the instance \mathcal{I} to give lower bounds. The first theorem says that any online algorithm transmitting only one packet per time step can not be optimal if it has an arbitrarily large additional buffer.

Theorem 1. *Let $k \geq 0$ be an integer. For any online algorithm A with a buffer of size $A = kB$ and a transmission rate of k times the adversary’s one, the competitive ratio of A is at least $\frac{1}{k+1}$.*

First, the adversary presents a sequence of the instance \mathcal{I} which makes A ’s buffer full. Before the first time step, i.e., in time interval $(0, 1)$, the adversary presents the instance \mathcal{I} . For the first B packets of weight one, any optimal online algorithm A must accept all the packets, because if it rejects a packet, then the adversary gives no packet any more and it accepts all the given packets. Then the adversary rejects the B packets of weight one and it accepts the next B packets of weight α . If $k \leq 1$, then A accepts at most B packets of weight α and its buffer is full. If $k > 1$, then A accepts B packets of weight α and it has free spaces in its buffer. Then during B time steps, no packet arrives and B packets in the buffer are transmitted. The adversary has the empty buffer and A contains B packets in its buffer. Again the adversary presents the instance \mathcal{I} and we can repeat the above argument. Each instance \mathcal{I} cause A to accumulate B packets in its buffer. So after some time steps, A ’s buffer is full.

After causing A ’s buffer full, the adversary gives no packet while B packets are transmitted. The adversary’s buffer is empty and A has B empty spaces in its buffer. Before the next transmission, the adversary presents the instance \mathcal{I} . Then accepting the first B packets of weight one, A has the full buffer and so it misses the B packets of weight α . But the adversary accepts all the packets of weight α . Until the gain of A is less than that of the adversary, we can repeat the argument. □

Theorem 2. *Let $s \geq 1$ be an integer. For any online algorithm A with a buffer of size $A = kB$ and a transmission rate of s times the adversary’s one, the competitive ratio of A is at least $k \geq 1 - \frac{1}{\alpha}$.*

In time interval $(0, 1)$, the adversary presents the instance \mathcal{I} . As in, any optimal online algorithm A accepts B packets of weight one. Then the adversary accepts the next B packets of weight α rejecting the B packets of weight one. So A should accept at least $(1 - \frac{1}{\alpha})B$ packets of the packets of weight α . That is, A should have additional buffer of at least size $(1 - \frac{1}{\alpha})B$. It is regardless of the transmission rate, because A should determine whether to accept or reject before the transmission. \square

Theorem 3. *Let \mathcal{A} be an optimal online algorithm with additional buffer $A = (1 - \frac{1}{\alpha})B$ and having a transmission rate being the s -fold of the adversary's one, where $s \geq 2$.*

Assume AL is an optimal online algorithm with additional buffer $A = (1 - \frac{1}{\alpha})B$ and having a transmission rate being the s -fold of the adversary's one, where $s < 2$. Then in time interval $(0, 1)$, the adversary presents the instance \mathcal{I} . Then AL accepts B packets of weight one and A packets of weight α . Also the buffers of both AL and the adversary are full. At time step one, s packets and one packet are transmitted by AL and the adversary, respectively. Then before the next transmission, a packet p of weight one arrives. AL should accept p , because if it rejects, then no packet arrive any more and the adversary accepts p . Next, a packet of weight α follows. Then AL can not accept the packet, because there is not enough space in its buffer. But the adversary rejects p and accept the packet of weight α . It is a contradiction. \square

3 Nonpreemptive Buffering Policy

3.1 Two Values Packets

In this subsection, we consider the case the weights of packets are given as only two values 1 and $\alpha > 1$, representing low and high priority packets, respectively. A packet of weight one is called an 1-packet and a packet of weight α an α -packet. First, we investigate an online policy with additional buffer $A = B$. Its performance is compared with that of the optimal offline policy with a buffer of size B , denoted by \mathcal{A}^* . We consider an abstract buffer model, where there are two buffers of each size B and at each integral time, the front packet of one of the buffers is transmitted. An online policy \mathcal{A} is devised on the model. It contains only α -packets in one buffer, called α -buffer, and only 1-packets in the other buffer, called 1-buffer. When a packet arrives, it is accepted if its corresponding buffer has a free space. The online policy \mathcal{A} always transmits a packet of α -buffer if it is not empty. When α -buffer is empty, it transmits a packet of 1-buffer.

Theorem 4. *Let \mathcal{A} be an online policy with additional buffer $A = B$ and having a transmission rate being the s -fold of the adversary's one, where $s \geq 2$.*

We show that at any time, the total number of packets in \mathcal{A} 's buffers is at least that of packets in \mathcal{A}^* 's buffer. It says that the number of packets transmitted by \mathcal{A} is at least that of packets transmitted by \mathcal{A}^* . Assume that

at some time, the total number of packets in \mathcal{A} 's buffers is less than in \mathcal{B} 's buffer. Let t be the first such time. Then immediately before t , say at t_0 , the number of packets in both buffers is same. At time t , a packet arrives, and \mathcal{A} rejects it while \mathcal{B} accepts. Since \mathcal{B} can accept the packet, there is a free space in its buffer at t_0 . So at t_0 , \mathcal{A} also contains packets less than B , and has free spaces in both α -buffer and 1-buffer. Thus \mathcal{A} can accept the packet regardless of its weight. It is a contradiction.

Now, we prove that all α -packets accepted by \mathcal{A} are also accepted by \mathcal{B} . Assume that an α -packet is accepted by \mathcal{A} but rejected by \mathcal{B} . Let p be such a packet that arrives earliest. Let \mathcal{E} be the set of α -packets which have arrived before p and are accepted by \mathcal{A} . All packets in \mathcal{E} are also accepted by \mathcal{B} . Note that the packets in \mathcal{E} are transmitted by \mathcal{A} earlier than by \mathcal{B} , because \mathcal{A} gives the priority of transmission to the α -buffer. Thus when p arrives, the number of packets in α -buffer of \mathcal{A} is at most that of packets in \mathcal{B} 's buffer. Since \mathcal{A} accepts p , \mathcal{B} also accepts it. It is a contradiction. Consequently, more packets are transmitted by \mathcal{A} than by \mathcal{B} and also more α -packets are transmitted by \mathcal{A} than by \mathcal{B} . Thus the gain of \mathcal{A} is at least that of \mathcal{B} . \square

We can develop an online policy \mathcal{A} with additional buffer $A = B$ to simulate the abstract policy \mathcal{B} . The \mathcal{A} makes a decision to accept or reject the packets based on the policy \mathcal{B} . That is, \mathcal{A} follows the decisions of \mathcal{B} . It is possible, because both policies have the same size buffer and transmit one packet per unit time. In nonpreemptive model, the throughput is determined only by the admission phase. So both policies have the same throughput.

Theorem 5. *Let \mathcal{A} and \mathcal{B} be any two policies with $A = B$.*

In the following, we shall try to reduce the size of buffer, preserving the optimality. The lower bound of the buffer size for the optimal online policy is shown. Also we show that matching the lower bound needs to increase the transmission rate. We consider a specific instance \mathcal{I} which the adversary present. In \mathcal{I} , all packets arrive during a short period, $(0, 1)$, before the next transmission time. First, B 1-packets arrive. And then B α -packets follow. In the next theorems, the adversary uses the instance \mathcal{I} to give lower bounds.

Theorem 6. *Let \mathcal{A} be any policy with $s \geq 1$. Let \mathcal{B} be any policy with $A = kB$ and $k \geq 1 - \frac{1}{\alpha}$.*

In time interval $(0, 1)$, the adversary presents the instance \mathcal{I} . Any optimal online policy \mathcal{B} should accept the B 1-packets, because if it rejects one of the packets, then no packet arrives any more later and its optimality is broken. Then the adversary accepts the next B α -packets rejecting the B 1-packets. So \mathcal{A} should accept at least $(1 - \frac{1}{\alpha})B$ α -packets. That is, A should have additional buffer of size at least $(1 - \frac{1}{\alpha})B$. It is regardless of the transmission rate, because A should determine whether to accept or reject the given packets of \mathcal{I} before the first transmission. \square

Theorem 7. *Let \mathcal{A} be an online policy with additional buffer $A = (1 - \frac{1}{\alpha})B$ and a transmission rate being s -fold of the adversary's one, where $s \geq 2$.*

Assume \mathcal{A} is an optimal online policy with additional buffer $A = (1 - \frac{1}{\alpha})B$ and having a transmission rate being the s -fold of the adversary's one, where $s < 2$. Then in time interval $(0, 1)$, the adversary presents the instance \mathcal{I} . Then \mathcal{A} accepts B 1-packets and A α -packets. Also both buffers of \mathcal{A} and OPT , respectively. Before the next transmission, an 1-packet p arrives. Then \mathcal{A} should accept p , because if \mathcal{A} rejects it, then no packet arrives any more and the adversary accepts p . Subsequently, an α -packet follows. Then AL cannot accept the packet, because there is not enough space in its buffer. But the adversary rejects p and accepts the α -packet. It is a contradiction. \square

Here we shall present an optimal online policy achieving the lower bounds given in the above theorems. For convenience of analysis, we can assume that the transmissions of packets by the online policy result in a busy period. That is, at each transmission time except the last, the online policy transmits exactly two packets and at the last time, it transmits at least one packet. Note that the online policy has a transmission rate being 2-fold of the adversary's one. Actually, we divide the whole schedule of online policy into busy periods, and we can analyze the performance of the policy independently for packets arriving in each busy period. Now we describe an online policy \mathcal{A} which has additional buffer $A = (1 - \frac{1}{\alpha})B$ and a transmission rate being the 2-fold of the adversary's one. It consists of phases and a phase proceeds until its buffer is full. The online policy \mathcal{A} maintains three variables l , h , and c . The 0-phase starts at time 0 and initially, l is set to B , h to 0, c to $\frac{2\alpha-1}{\alpha}B$. The 0-phase ends when the buffer is full. Actually, at time t when a packet arrives, the buffer is full and the 0-phase proceeds until time $\lceil t \rceil$, the next transmission time. At time $\lceil t \rceil$, the 1-phase starts and it continues until the next time when the buffer is full. In the i -phase ($i \geq 1$), \mathcal{A} behaves differently from in the 0-phase. When the i -phase starts, the variables are reset as follows: $l = 1$, $h = 0$, and $c = 2$. When an 1-packet arrives, \mathcal{A} accepts the packet if $l + 1 \leq c$ and then it increases the value of l one. Also when an α -packet arrives, \mathcal{A} accepts it if there is an empty space in the buffer. After each transmission, l , each integral time, l increases the value of l one and the value of c two. Specifically, at each time when a packet arrives, 1-packet is accepted if the total number of accepted 1-packets containing itself is at most l , and α -packet is greedily accepted if there is an empty space in the buffer. The value of c represents the total number of spaces which \mathcal{A} provides to accept the packets in a phase. Since \mathcal{A} has a transmission rate being the 2-fold of the adversary's one, c is increased two after each transmission and also when an i -phase ($i \geq 1$) starts, c is set to two. The value of l represents the total number of 1-packets currently accepted in a phase and the value of c restricts that of l less than or equal to its value.

Theorem 8. *Let $\alpha \geq 1$ and $s \geq 1$ be integers. Let $A = (1 - \frac{1}{\alpha})B$ be the buffer size of the adversary.*

First we consider the case the buffer is never full at any time. Then there is only 0-phase. We can see that all α -packets accepted by \mathcal{A} are also accepted by \mathcal{B} , because \mathcal{B} greedily accepts α -packets if there is an empty space in the buffer. Also after any k -th transmission, at most $(B + k)$ 1-packets are accepted by \mathcal{A} . But the buffer size of \mathcal{B} is set to $B + k$ and so if an 1-packet accepted by \mathcal{A} is rejected by \mathcal{B} , then $(B + k)$ 1-packets are already accepted by \mathcal{B} . So \mathcal{B} accepts 1-packets at least as many as 1-packets accepted by \mathcal{A} .

Next, we assume that the buffer is full at some time in the processing of \mathcal{A} . Let t_0 be the first time when the buffer is full. Then the 0-phase ends at time $\lceil t_0 \rceil$, say k_0 . Also we see that \mathcal{A} has $B + k_0 - 1$ spaces, \mathcal{B} has $\frac{2\alpha - 1}{\alpha}B + 2(k_0 - 1)$. Since the buffer is full at t_0 , \mathcal{A} accepts the packets to fill all the spaces of \mathcal{A} . But at most $\frac{2\alpha - 1}{\alpha}B + 2(k_0 - 1)$ 1-packets of these packets are accepted. So the gain of \mathcal{B} for the packets accepted in the 0-phase is at least $\frac{2\alpha - 1}{\alpha}B + \alpha(\frac{2\alpha - 1}{\alpha}B + 2(k_0 - 1)) = \alpha B + (\alpha + 1)(k_0 - 1)$. Also at most $(B + k_0 - 1)$ spaces which can accommodate packets are made by \mathcal{A} in the 0-phase. So the gain of \mathcal{A} is at most $\alpha(B + k_0 - 1)$ even if it contains only the α -packets in the spaces made in the 0-phase. (Here note that in \mathcal{A} , the packets contained in the spaces made in the 0-phase may arrive in the next phase.) Thus the gain of \mathcal{B} is at least that of \mathcal{A} in the 0-phase. Now, we consider the i -phase ($i \geq 1$) except the last phase. Let k_i be the time when the i -phase ends. Then we can see that \mathcal{A} has $k_i - k_{i-1}$ spaces, \mathcal{B} has $2(k_i - k_{i-1})$. Since \mathcal{A} accepts exactly $k_i - k_{i-1}$ packets and at most $2(k_i - k_{i-1})$ packets of them are 1-packets, the gain of \mathcal{B} is at least $(\alpha + 1)(k_i - k_{i-1})$. But at most $k_i - k_{i-1}$ new spaces are made by \mathcal{A} in the i -phase, and the gain of \mathcal{A} increases at most $\alpha(k_i - k_{i-1})$. So \mathcal{B} also has the gain at least as much as \mathcal{A} in the i -phase. For the last phase, the case is similar to the above one that the buffer is never full at any time. \square

3.2 Arbitrary Values Packets

In this subsection, we consider the case that packets have arbitrary weights. The lowest weight is assumed to be one and the largest weight α is known. First we show that any optimal online policy should have additional buffer of at least $\Omega((\log \alpha)B)$. Also to match the lower bound of buffer size, it is shown that any online policy needs to have a transmission rate being at least the $\log \alpha$ -fold of the adversary's one.

Theorem 9. *Let $\alpha \geq 1$ and $s \geq 1$ be integers. Let $A = kB$ be the buffer size of the adversary. Let $k \geq \frac{1}{2} \log \alpha$ be the transmission rate of the adversary.*

All packets of the instance arrive during time interval $(0, 1)$. Let ℓ be an integer such that $2^\ell = \alpha$. First, B packets of weight one arrive. The packets should be all accepted by any optimal online policy \mathcal{B} as before. Next, B packets of weight two arrive. Then \mathcal{B} should accept at least $\frac{B}{2}$ packets of them

to preserve the optimality. Specifically, for each $j = 1, \dots, \ell$, B packets of weight 2^j are given by the adversary and \mathcal{A} accepts at least $\frac{B}{2}$ packets of those ones, because $B + B \sum_{i=1}^j 2^{i-1} = 2^j B$. So the size of \mathcal{A} 's buffer should be at least $B + \ell \frac{B}{2} = (1 + \frac{\log \alpha}{2})B$. \square

Theorem 10. *Let \mathcal{A} be an online policy with a buffer of size $A = (\frac{1}{2} \log \alpha)B$. Then, for any adversary \mathcal{A} , \mathcal{A} achieves a competitive ratio of at least $s \geq \log \alpha$.*

During time interval $(0, 1)$, the adversary presents the same packets as in Theorem 9. The buffer of any optimal online policy \mathcal{A} becomes full. Also the \mathcal{A} 's buffer contains B packets of weight α . After the first transmission, \mathcal{A} 's buffer has s empty spaces and \mathcal{A} 's buffer one empty space. A sequence of packets arrives before the next transmission. For each $j = 1, \dots, \ell$ ($2^\ell = \alpha$), if \mathcal{A} accepts the given packets of weight from 2 to 2^{j-1} , then a packet of weight 2^j arrives. Since $\sum_{i=1}^{j-1} 2^i < 2^j$, \mathcal{A} should accept the packet of weight 2^j to preserve the optimality. Thus \mathcal{A} must have empty spaces of at least ℓ , that is, the transmission rate of \mathcal{A} is at least ℓ . \square

We will establish an optimal online policy asymptotically matching the lower bounds given in Theorem 9 and 10. Like the online policy P introduced previously, an artificial buffer model is considered, where there are totally $(\lceil \log \alpha \rceil + 1)$ buffers of size $2B$ and at each integral time, the front two packets from each buffer are simultaneously transmitted. We derive an online policy \mathcal{A} on the model as follows: Each buffer is denoted by i -buffer, $i = 0, \dots, \lceil \log \alpha \rceil$. In \mathcal{A} , a packet of weight w is stored in i -buffer if $2^i \leq w < 2^{i+1}$. When a packet arrives, \mathcal{A} greedily accepts it if there is an empty space in its corresponding buffer.

Theorem 11. *Let \mathcal{A} be an online policy with a buffer of size $A = (\frac{1}{2} \log \alpha)B$. Then, for any adversary \mathcal{A} , \mathcal{A} achieves a competitive ratio of at least $s \geq \log \alpha$.*

We call a packet of weight w an i -packet if $2^i \leq w < 2^{i+1}$. Then an i -packet is stored in i -buffer in \mathcal{A} . We consider an assignment of i -packets accepted by \mathcal{A} to i -packets accepted by \mathcal{A} . When an i -packet arrives, if it is accepted by both \mathcal{A} and \mathcal{A} , then it is assigned to itself. If it is rejected by \mathcal{A} but accepted by \mathcal{A} , then the i -buffer of \mathcal{A} is full, and two i -packets not yet assigned in the buffer are assigned to the packet. For convenience, we reorder packets in buffers of \mathcal{A} as follows: The assigned packets are moved ahead such that a sequence of assigned packets is located in the front of the buffer at any time. Then we show that at any time and for any i -buffer of \mathcal{A} , the number of assigned packets in the i -buffer is at most two times the number of i -packets in the buffer of \mathcal{A} . It is shown by induction on time. Assume that until time t , it is true. At the next time, one of two events occurs. If a transmission occurs, then the number of assigned packets in i -buffer decreases two and the number of i -packets in \mathcal{A} 's buffer decreases at most one. If an i -packet arrives and it is accepted by \mathcal{A} , then there are two cases; it is also accepted by \mathcal{A} or rejected by \mathcal{A} . For the former

case, the number of assigned packets in i -buffer increases one and the number of i -packets in σ_i 's buffer increase one. For the later case, the i -buffer of σ_i is full and by the assumption of induction, there are at least two unassigned packets in the i -buffer. So two i -packets in i -buffer can be newly assigned to. Thus the number of assigned packets in i -buffer increases two and the number of i -packets in σ_i 's buffer increase one. For all the cases, the invariant is maintained. It directly says that the assignment is well-defined. Also the weight of any packet accepted by σ_i is at most the total weight of packets assigned to it. So the gain of σ_i is at most that of σ_i . \square

We also develop an online policy σ_{α} with a buffer of size $2(\lfloor \log \alpha \rfloor + 1)B$ and a transmission rate being the $2(\lfloor \log \alpha \rfloor + 1)$ -fold of the adversary's one which simulates the policy σ . The σ_{α} makes a decision to accept or reject the packets based on σ . Since both policies have buffers of the same size and transmit the same number of packets per unit time, they have a same throughput. So the online policy σ_{α} is optimal versus the offline adversary.

Theorem 12. *Let σ be an α -competitive algorithm with a buffer of size $2(\lfloor \log \alpha \rfloor + 1)B$ and a transmission rate being the $2(\lfloor \log \alpha \rfloor + 1)$ -fold of the adversary's one. Then σ_{α} is an α -competitive algorithm with a buffer of size $2(\lfloor \log \alpha \rfloor + 1)B$ and a transmission rate being the $2(\lfloor \log \alpha \rfloor + 1)$ -fold of the adversary's one.*

4 Preemptive Buffering Policy

In this section, we are concerned with the preemptive model, where the buffering policy can drop packets already accepted in the buffer, in contrast to the non-preemptive model discussed previously. Specifically, we assume that dropping packets is allowed only when the buffer is full, as in [8]. Packets have arbitrary positive weights, and we investigate the online policy $\sigma_{s,k}$ introduced in [8]. The authors showed that $\sigma_{s,k}$ is 2-competitive. Here the policy $\sigma_{s,k}$ is allowed to have additional buffer $A = kB$ and a transmission rate being the s -fold ($s \geq 2$) of the adversary's one. The size A of additional buffer depends on the value of s such that the $\sigma_{s,k}$ is optimal. In $\sigma_{s,k}$, an arriving packet is accepted if there is an empty space in the buffer. Otherwise, that is, if buffer overflow occurs, then a packet with a lowest weight among packets currently contained in the buffer is selected and its weight is compared with that of the arriving packet. The packet with the lower weight is dropped.

We show that the policy $\sigma_{s,k}$ is optimal if $k = \frac{1}{s-1}$ for a given $s \geq 2$. The basic idea of its proof is based on the proof given in [8].

Theorem 13. *Let σ be an α -competitive algorithm with a buffer of size $A = kB$ and a transmission rate being the s -fold of the adversary's one, where $k = \frac{1}{s-1}$ and $s \geq 2$.*

For convenience, we can assume that the transmissions of the σ result in one busy period, where except the last time, exactly s packets are transmitted at each integral time. Let \mathcal{D} be the set of packets which are transmitted by σ but dropped by $\sigma_{s,k}$ and let \mathcal{G} be the set of packets which

are transmitted by \mathcal{G} but dropped by \mathcal{D} . Then we construct a mapping m from \mathcal{D} to \mathcal{G} as follows: The packets in \mathcal{D} are mapped in the order of their dropping times. Then a packet p in \mathcal{D} is mapped to a packet $m(p)$ in \mathcal{G} transmitted earliest among packets not yet mapped to. We prove that for each packet p in \mathcal{D} , the weight of p is at most that of $m(p)$. It is sufficient to prove that the algorithm is optimal.

We show that by induction on time when a packet in \mathcal{D} is dropped. For the basis of induction, if a packet p in \mathcal{D} is the first packet to be dropped at time t , then it is mapped to the packet transmitted at time $\lceil t \rceil$, the weight of which is at least that of p , because at time t , the buffer is full and until $\lceil t \rceil$, all the packets in the buffer have weights more than the weight of p .

Assume that a packet p in \mathcal{D} is dropped at time t and all packets in \mathcal{D} which are dropped at time $s < t$ are already mapped to packets in \mathcal{G} having weights more than their ones. Let $r(t)$ be the latest time before or at time t such that no packets in \mathcal{D} dropped before $r(t)$ are mapped to packets transmitted at or after $r(t)$. If no such time exists, then $r(t) = 0$. Let $s(t)$ be the earliest time at or after time t such that the buffer is full at $s(t)$ and all packets contained in the buffer at $s(t)$ are transmitted. Such time $s(t)$ exists, because the buffer is also full at t . Let $\bar{s}(t)$ be $\lceil s(t) \rceil$. The packets contained in the buffer at $s(t)$ are transmitted during time interval $[\bar{s}(t), \bar{s}(t) + kB]$, because $\frac{1+k}{s} = k$.

We claim that the weights of packets transmitted in time interval $[t, \bar{s}(t) + kB]$ are at least the weight of p , denoted by $w(p)$. Since the buffer is full at time t and the weights of packets contained in the buffer at t are at least $w(p)$, the weights of packets transmitted during time interval $[[t], [t] + kB]$ are also at least $w(p)$. If $s(t) = t$, then the claim is true. Otherwise, $s(t) > t$, let $t' \in [[t], [t] + kB]$ be the earliest time when a packet being in the buffer at time t is dropped. Then the weights of packets transmitted in $[[t'], [t'] + kB]$ are also at least $w(p)$. If $s(t) = t'$, then the claim is true. Otherwise, continue the same argument.

We will complete the proof by proving that the packet p is mapped to a packet in \mathcal{G} transmitted during $[t, \bar{s}(t) + kB]$. Let \mathcal{I} be the set of packets transmitted by \mathcal{G} which are also transmitted by \mathcal{D} during $[r(t), \bar{s}(t) + kB]$ and let $\mathcal{D}[u, v]$ be the set of packets in \mathcal{D} dropped during time interval $[u, v]$. If a packet is in \mathcal{I} or $\mathcal{D}[r(t), s(t)]$, then it arrives in $[r(t) - kB, s(t)]$, because no packet in \mathcal{I} arrives during $(s(t), \bar{s}(t) + kB]$ from the definition of $s(t)$. Thus all packets in \mathcal{I} or $\mathcal{D}[r(t), s(t)]$ may be transmitted by \mathcal{G} in $[r(t) - kB, \bar{s}(t) + B]$. So we can see that $|\mathcal{D}[r(t), s(t)]| + |\mathcal{I}| \leq \bar{s}(t) - r(t) + (k + 1)B$. On the other hand, we consider all packets transmitted by \mathcal{G} during $[r(t), \bar{s}(t) + kB]$, to which no packets in \mathcal{D} dropped before $r(t)$ are mapped. The total number of such packets is at least $s(\bar{s}(t) + kB - r(t)) = s(\bar{s}(t) - r(t)) + skB$. Since $sk = k + 1$, the number of packets transmitted by \mathcal{G} during $[r(t), \bar{s}(t) + kB]$ is at least $|\mathcal{D}[r(t), s(t)]| + |\mathcal{I}|$. It implies that the number of packets in \mathcal{G} transmitted during $[r(t), \bar{s}(t) + kB]$ is at least $|\mathcal{D}[r(t), s(t)]|$. After packets in \mathcal{D} dropped during $[r(t), t)$ are mapped, that is, when the packet p is mapped, the number of packets in \mathcal{G} transmitted during $[r(t), \bar{s}(t) + kB]$ and not yet mapped to is at least $|\mathcal{D}[t, s(t)]|$. Also from the definition of $r(t)$, all such packets not yet mapped to are transmitted after t .

Thus there is at least one packet in \mathcal{G} which p can be mapped to and the packet is transmitted during $[t, \bar{s}(t) + kB]$, that is, its weight is more than $w(p)$. \square

References

- [1] W.A. Aiello, Y. Mansour, S. Rajagopalan, and A. Rosen. Competitive queue policies for differentiated services. In *Proc. of the IEEE INFOCOM*, pages 431–440, 2000.
- [2] S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. In *Proc. of 36th ACM Symposium on Theory of Computing*, to appear, 2004.
- [3] Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. In *Proc. of 35th ACM Symposium on Theory of Computing*, pages 82–89, 2003.
- [4] A. Bar-Noy, A. Freund, S. Landa, and J. Naor. Competitive on-line switching policies. In *Proc. of 13th ACM Symposium on Discrete Algorithms*, pages 525–534, 2002.
- [5] M. Brehob, E. Torng, and P. Uthaisombut. Applying extra-resource analysis to load balancing. In *Proc. of 11th ACM Symposium on Discrete Algorithms*, pages 560–561, 2000.
- [6] R. Fleischer and H. Koga. Balanced scheduling toward loss-free packet queuing and delay fairness. *Algorithmica*, 38:363–376, 2004.
- [7] B. Kalyanasundaram and K.R. Pruhs. Speed is as powerful as clairvoyance. *J. of ACM*, 47(4):617–643, 2000.
- [8] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. of 31th ACM Symposium on Theory of Computing*, pages 520–529, 2001.
- [9] A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proc. of 11th European Symposium on Algorithms*, pages 361–372, 2003.
- [10] C.Y. Koo, T.W. Lam, J. Ngan, and K.K. To. Extra processors versus future information in optimal deadline scheduling. In *Proc. of 14th ACM Symposium on Parallel Algorithms and Architectures*, pages 133–142, 2002.
- [11] T.W. Lam and K.K. To. Performance guarantee for online deadline scheduling in the presence of overload. In *Proc. of 12th ACM Symposium on Discrete Algorithms*, pages 755–764, 2001.
- [12] Z. Lotker and B. Patt-Shamir. Nearly optimal FIFO buffer management for Diff-Serv. In *Proc. of 21th ACM Symposium on Principles of Distributed Computing*, pages 134–142, 2002.
- [13] C.A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proc. of 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.
- [14] A. Zhu. Analysis of queueing policies in QoS switches. *J. of Algorithms*, to appear.

Oriented Paths in Mixed Graphs

Egon Wanke¹ and Rolf Kötter²

¹ Institute of Computer Science,

² C. & O. Vogt Brain Research Institute, and ²Institute of Anatomy II,
Heinrich-Heine-Universität, D-40225 Düsseldorf, Germany

Abstract. We consider mixed graphs with directed and undirected edges. A path in a mixed graph is called oriented if it has at least one directed edge. We show that 1.) oriented paths can be found in polynomial time, 2.) computing a maximal number of mutually edge-disjoint oriented s, t -paths is NP-complete, and 3.) computing a minimal set of edges or vertices whose removal destroys all oriented s, t -paths is NP-complete. In mixed graphs, the gap between the maximal number of mutually edge-disjoint oriented s, t -paths and the minimal number of edges or vertices in an s, t -cut can be arbitrary large. Finally we introduce simple 2-approximation algorithms for computing vertex and edge s, t -cuts.

1 Motivation and Introduction

This work is motivated by the analysis of different parcellation schemes of the brain. A parcellation scheme divides the cerebral cortex into a set of disjoint brain structures. These brain structures are regarded as elementary units that form the basis for localizing all structural and functional characteristics. Comparisons between data from different experimental studies have always been difficult due to the incompatibility of the many parcellation schemes used. The different parcellation schemes result mainly from inter-individual differences, methodological ambiguities, and observer-dependent criteria [Ess85, Zil04]. Therefore, different maps often differ considerably in the areal boundaries and the nomenclature of the areas. The same name may even be given to areas that are only partially coextensive, and for many areas their relationships remain unclear.

A first systematic approach to this parcellation problem was the Objective Relational Transformation of Stephan, Zilles and Kötter [SZK00]. This method enables databases to handle the huge variability of parcellation schemes allowing a conversion of the data from incongruent maps objectively and reproducibly into any user-chosen parcellation scheme. It is based on a classification of the relations between pairs of areas in different maps as identity, a strict inclusion, or a real overlap. To infer new knowledge from existing knowledge, all the information about the relations between different maps is stored in a graph whose vertices represent the areas. Two vertices are connected by an edge if there is a known relation between the two corresponding areas. New information is obtained by following a path from area A along the known relations to some area B . For example, a strict inclusion of area A into area B is obtained by a path from A to B along edges which only represent identities and strict inclusions, but at least

one strict inclusion. We are mainly interested in simple paths, where an area is passed at most once. In a graph theoretical context, this corresponds to finding simple oriented paths in mixed graphs, where the undirected edges represent identities and the directed edges represent strict inclusions.

A major additional problem is the logical inconsistency of the available data arising from the several sources of error. Thus it is not unusual that there are paths between two areas of which some represent identities and some represent strict inclusions. This problem has been addressed by introducing Precision of Description Codes (PDCs), which classify the quality of the evidence for the relation being provided. These PDCs can be regarded as weights attached to edges of the graph. Where several statements exist concerning the same edge, the corresponding set of PDCs can be used in more complex weightings.

From the neuro science point of view the aims are 1.) to find as many paths as possible to allow a conversion of information, 2.) to decide among alternative paths based on their classification and PDC weightings, 3.) to perform the conversion of information efficiently. Here we consider graph theoretical methods to achieve these goals.

This paper is organized as follows. In Section 2, we introduce the basic definitions for general, edge-simple, and vertex-simple mutually edge-disjoint and mutually vertex-disjoint paths in mixed graphs. In Section 3, we show that general, edge-simple, and vertex-simple oriented paths in mixed graphs with n vertices and m edges can be found in time $O(n + m)$, $O(n \cdot m^2)$, and $O(n^2 \cdot m)$, respectively. In Section 4, we show the NP-completeness of the k oriented mutually disjoint paths problem for the following cases.

	general	edge-simple	vertex-simple
edge-disjoint	case 1: $k \geq 2$	case 3: k unbounded	case 5: $k \geq 2$
vertex-disjoint	case 2: $k \geq 2$	case 4: $k \geq 2$	case 6: k unbounded

In Section 5, we consider minimal sets of edges and vertices such that a mixed graph without these edges and vertices, respectively, has no oriented path between two vertices s and t . We prove that minimal s, t -cut set problems are NP-complete. The gap between the size of a minimal s, t -cut set and a maximal number of oriented s, t -paths can be arbitrary large for general mutually vertex-disjoint and vertex-simple mutually edge-disjoint oriented paths. For edge-simple mutually edge-disjoint and general mutually edge-disjoint oriented paths, this gap is at least 3 and 2, respectively. Conclusions, discussions, and simple polynomial time 2-approximation algorithms for computing vertex and edge s, t -cuts are given in Section 6. This is the first work in literature which considers oriented paths in mixed graphs.

2 Preliminaries

A *mixed graph* $G=(V, A, E)$ consists of a finite set of vertices V , a set of directed edges $A \subseteq V \times V$, and a set of undirected edges $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. It is called *undirected* or *directed* if it has only undirected or only directed edges,

respectively. A mixed graph $J = (V', A', E')$ is a *subgraph* of G if $V' \subseteq V$, $A' \subseteq A$, and $E' \subseteq E$. It is an *induced subgraph* of G if additionally $A' = \{(u, v) \in A \mid u, v \in V'\}$ and $E' = \{\{u, v\} \in E \mid u, v \in V'\}$.

Let $p = u_0, e_1, u_1, \dots, e_k, u_k$ be an alternating sequence of $k + 1$ vertices $u_0, \dots, u_k \in V$ and k edges $e_1, \dots, e_k \in A \cup E$. Sequence p is called a *path* of length k if $e_i = (u_{i-1}, u_i)$ for all directed edges e_i of p and $e_i = \{u_{i-1}, u_i\}$ for all undirected edges e_i of p . A mixed path is called *undirected*, *directed*, or *mixed* if it contains only undirected edges, only directed edges, or at least one directed edge, respectively.

In this paper, we study the following path problems. Given a mixed graph $G = (V, A, E)$ and two vertices $s, t \in V$.

1. Find an oriented path from s to t .
2. Compute the maximal number of mutually disjoint oriented s, t -paths.
3. Compute the minimal number of edges of $A \cup E$ or vertices of V such that G without these edges or vertices has no oriented s, t -path.

A mixed path $p = u_0, e_1, u_1, \dots, e_k, u_k$ is called *edge-simple* or *vertex-simple* if all edges e_1, \dots, e_k or all vertices u_0, \dots, u_k , respectively, are distinct. Two mixed paths $p = u_0, e_1, u_1, \dots, e_k, u_k$ and $q = u'_0, e'_1, u'_1, \dots, e'_{k'}, u'_{k'}$ are *edge-disjoint* if $\{e_1, \dots, e_k\} \cap \{e'_1, \dots, e'_{k'}\} = \emptyset$ and *vertex-disjoint* if $\{u_0, \dots, u_k\} \cap \{u'_1, \dots, u'_{k'-1}\} = \emptyset$ and $\{u_1, \dots, u_{k-1}\} \cap \{u'_0, \dots, u'_{k'}\} = \emptyset$. We analyze the complexities of the problems above for general, edge-simple, and vertex-simple oriented mutually edge-disjoint and mutually vertex-disjoint paths.

We first observe that k edge-simple oriented paths can always be found with an algorithm for finding k vertex-simple oriented paths applied to the k -line graph G^k of G defined as follows.

Definition 1. Let $G = (V, A, E)$ be a mixed graph. The k -line graph $G^k = (V^k, A^k, E^k)$ of G is defined by $V^k = A \cup E \cup \{(u, i) \mid u \in V, 1 \leq i \leq k\}$.

- $A^k = \{((u_1, u_2), (u_2, i)) \mid (u_1, u_2) \in A, 1 \leq i \leq k\} \cup \{((u_1, i), (u_1, u_2)) \mid (u_1, u_2) \in A, 1 \leq i \leq k\}$
- $E^k = \{\{(u_1, i), \{u_1, u_2\}\} \mid \{u_1, u_2\} \in E, 1 \leq i \leq k\}$

If G has n vertices and m edges then G^k has $m + k \cdot n$ vertices and $2 \cdot k \cdot m$ edges. Let s and t be two vertices of G and let $G_{s,t}$ be the graph G with two additional vertices s', t' and two additional undirected edges $\{s', s\}$ and $\{t', t\}$. Then the mixed graph G has k undirected, directed, or oriented mutually edge-disjoint s, t -paths if and only if the k -line graph $G^k_{s,t}$ of $G_{s,t}$ has k undirected, directed, or oriented mutually vertex-disjoint $\{s', s\}, \{t', t\}$ -paths. The k s, t -paths in G are edge-simple if and only if the corresponding k $\{s', s\}, \{t', t\}$ -paths in $G^k_{s,t}$ are vertex-simple. Thus, the k mutually edge-disjoint oriented paths problem can be solved in polynomial time by a polynomial time algorithm for the k mutually vertex-disjoint oriented paths problem.

3 Oriented Paths

A mixed s, t -path can obviously be found in linear time by Depth First Search. The problem of finding an oriented s, t -path in a mixed graph is only difficult and thus interesting if there is at least one undirected path between s and t . Otherwise, every mixed path from s to t is oriented. In this case, G has a mixed s, t -path if and only if it has an oriented s, t -path if and only if it has a vertex-simple oriented s, t -path. For the rest of this paper, we assume that there is always an undirected path between s and t , i.e. s and t are in the same connected component of the underlying undirected graph (V, E) of $G = (V, A, E)$.

The problem of finding an oriented s, t -path in a mixed graph G is equivalent to the problem of finding a mixed path from vertex $(s, 1)$ to vertex $(t, 2)$ in the two level graph G' defined as below. This shows that finding an oriented path can be done in linear time.

Definition 2. Let $G = (V, A, E)$ be a mixed graph. The two level graph $G' = (V', A', E')$ is defined by $V' = V \times \{1, 2\}$

$$A' = \{((u, 1), (v, 2)) \mid (u, v) \in A\} \cup \{((u, 2), (v, 2)) \mid (u, v) \in A\}$$

$$E' = \{(u, 1), (v, 1) \mid \{u, v\} \in E\} \cup \{(u, 2), (v, 2) \mid \{u, v\} \in E\}$$

However, an edge-simple (a vertex-simple) mixed $(s, 1), (t, 2)$ -path in the two level graph G' does not necessarily correspond to an edge-simple (a vertex-simple, respectively) oriented s, t -path in G . The example of Figure 1 (to the right) shows a mixed graph G whose two level graph G' has a vertex-simple $(v_2, 1), (v_1, 2)$ -path although G has no vertex-simple oriented v_2, v_1 -paths.

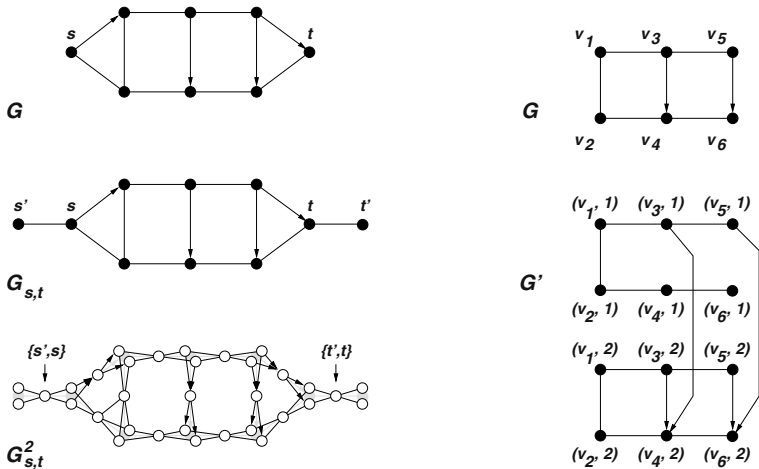


Fig. 1. To the left: A mixed graph G with two vertices s and t , the mixed graph $G_{s,t}$, and the 2-line graph $G_{s,t}^2$. To the right: A mixed graph G and the two level graph G'

If a mixed graph G contains only one directed edge (u, v) then finding an edge-simple or vertex-simple oriented s, t -path is equivalent to finding two edge-disjoint or vertex-disjoint undirected paths, respectively, in an undirected graph, one between s and u and one between v and t . The two vertex-disjoint paths problem for undirected graphs can be solved in time $O(n \cdot m)$, where n is the number of vertices and m is the number of edges, see [Shi80]. The two edge-disjoint paths problem for undirected graphs can be solved in time $O(m^2)$ by the 2-line graph construction as defined in Definition 1 and an algorithm for the two vertex-disjoint paths problem for undirected graphs.

We next consider the general case. Let $\tilde{G}^{s,t} = (V^{s,t}, \tilde{E}^{s,t})$, $\tilde{V}^{s,t} \subseteq V$, $\tilde{E}^{s,t} \subseteq E$, be the connected component of the underlying undirected graph of $G = (V, A, E)$ to which s and t belong. Let $p = u_0, e_1, u_1, \dots, e_k, u_k$ be an oriented path in G . We say path p connected component $\tilde{G}^{s,t}$ at vertex u_i , $0 \leq i \leq k-1$, if u_i belongs to $V^{s,t}$ and e_{i+1} is a directed edge. We say path p connected component $\tilde{G}^{s,t}$ at vertex u_i , $1 \leq i \leq k$, if u_i belongs to $\tilde{V}^{s,t}$ and e_i is a directed edge.

Lemma 1. $G = (V, A, E)$ s, t
 $\tilde{G}^{s,t} \subseteq G$ $(\tilde{V}^{s,t}, \tilde{E}^{s,t})$
 $s, t \in G$ $(\tilde{V}^{s,t}, \tilde{E}^{s,t})$
 $s, t \in G$ $\tilde{G}^{s,t}$

Let $p = s, e_1, u_1, \dots, e_l, t$ be an edge-simple (a vertex-simple) oriented s, t -path that leaves and enters $\tilde{G}^{s,t}$ more than once. Assume further path p leaves $\tilde{G}^{s,t}$ the first time at vertex u_{i_1} , enters $\tilde{G}^{s,t}$ the first time at vertex u_{i_2} , leaves $\tilde{G}^{s,t}$ the second time at vertex u_{i_3} , enters $\tilde{G}^{s,t}$ the second time at vertex u_{i_4} , and so on, see Figure 2.

Let q be any vertex-simple undirected path in $\tilde{G}^{s,t}$ from u_{i_2} to t . Let v be the first vertex of q that belongs either to the first part of p from s to u_{i_1} or to the last part of p from u_{i_4} to t . Such a vertex v always exists because t is the last vertex of p .

If v is from the first part of p from s to u_{i_1} , then let p' be the path that starts with the first part of p from s to v , continuous with the vertices and edges of q from v in reverse direction up to the first vertex w that belongs to the part of p from u_{i_2} to u_{i_3} , and continuous with the last part of p from w to t . If v is from the last part of p from u_{i_4} to t , then let p' be the path that starts with the first part of p from s to u_{i_2} , continuous with the part of q from u_{i_2} to v , and finishes with the last part of p from v to t . Note that in this case the last part of p could be empty ($v = t$). In both cases, we get an edge-simple (a vertex-simple) oriented path p' from s to t that leaves and enters the connected component $\tilde{G}^{s,t}$ less often than p . Thus, a simple inductive argumentation proves the lemma.

We use Lemma 1 to show that edge-simple and vertex-simple oriented paths can be found in polynomial time.

Theorem 1. $G = (V, A, E)$ n m
 $s, t \in G$ $(\tilde{V}^{s,t}, \tilde{E}^{s,t})$
 $s, t \in G$ $O(n \cdot m^2)$ ($O(n^2 \cdot m)$)

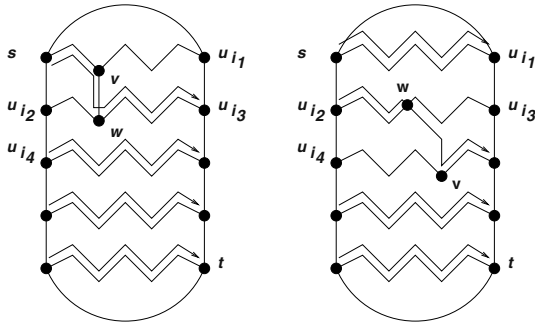


Fig. 2. Two possible cases for constructing a shorter oriented path p' from s to t . The left side illustrates the case where q passes first a vertex v of the part from s to u_{i_1} . The right side illustrates the case where q passes first a vertex v of the part from u_{i_4} to t

We can find an edge-simple (a vertex-simple) oriented path from s to t as follows. Compute for every vertex u of $\tilde{G}^{s,t} = (\tilde{V}^{s,t}, \tilde{E}^{s,t})$ all vertices v such that there is a path from u to v in mixed graph $G' = (V', A', E')$, where $V' = (V - \tilde{V}^{s,t}) \cup \{u, v\}$, $A' = A - \{(u, v) \in A \mid u \in V' \vee v \in V'\}$, and $E' = E - \tilde{E}^{s,t}$. Let G'' be the undirected graph $\tilde{G}^{s,t}$ with an additional vertex w and additional undirected edges between w and the vertices v as specified above. Next we compute two edge-disjoint (vertex-disjoint) paths between s, u and t, w in G'' . These two undirected paths exist for some u if and only if there is an edge-simple (a vertex-simple) path from s to t that leaves and enters $\tilde{G}^{s,t}$ exactly once. By Lemma 1, we know that such a path exists if and only if there is an edge-simple (a vertex-simple) oriented path from s to t in G . The overall running time is $O(n \cdot m^2)$ ($O(n^2 \cdot m)$), because two edge-disjoint (vertex-disjoint) paths in undirected graphs can be found in time $O(m^2)$ ($O(n \cdot m)$, respectively) and all the remaining constructions take linear time for every vertex u .

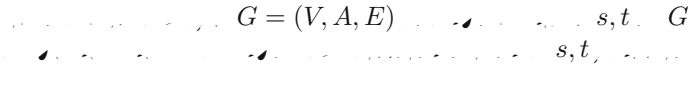
4 Disjoint Oriented s, t -Paths

Let us first discuss general paths, which are not necessarily oriented. Finding k edge-simple mutually edge-disjoint directed s, t -paths in a directed graph G can be done in polynomial time by using network flow technics, see for example, [AMO93]. In a mixed graph G , k mutually edge-disjoint s, t -paths can be found as follows. First substitute every undirected edge $\{u_{i-1}, u_i\}$ by two directed edges (u_{i-1}, u_i) and (u_i, u_{i-1}) and then compute a maximal number of edge-simple edge-disjoint s, t -paths in the resulting directed graph G' . If there are two s, t -paths p and q in G' such that p has a directed edge (u, v) and q has the reverse directed edge (v, u) then replace p and q by p' and q' as follows. Let p' be the path consisting of the first part of p from s to u and the last part of q from u to t , and let q' be the path consisting of the first part of q from s to v and the last part

of p from v to t . After all possible replacements the resulting k paths represent k edge-simple mutually edge-disjoint paths in the original mixed graph G .

k vertex-simple mutually vertex-disjoint paths in mixed graphs can also be found in polynomial time. Substitute in the directed graph G' defined above every vertex u_i by two vertices $u_{i,1}$ and $u_{i,2}$ and a directed edge $(u_{i,1}, u_{i,2})$, and every directed edge (u_i, u_j) by directed edge $(u_{i,2}, u_{j,1})$, as shown in Figure 3 on the left side. (The vertices s and t are replaced by s_1, s_2 and t_1, t_2 , respectively.) Then there are k vertex-simple mutually vertex-disjoint s, t -paths in the original mixed graph G if and only if there are k edge-simple mutually edge-disjoint s_2, t_1 -paths in the modified graph G' .

Now we consider oriented paths in mixed graphs.

Theorem 2.  $G = (V, A, E)$ \rightarrow $G' = (V', A', E')$ s, t G s_2, t_1 G'

The problem obviously belongs to NP. Let $H = (V, A)$ be a directed graph and $s_1, t_1, s_2, t_2 \in V$ be 4 distinct vertices of H . The problem to decide whether there are two vertex-disjoint paths in a directed graph H , one from s_1 to t_1 and one from s_2 to t_2 , is known to be NP-complete, see [FHW80]. By the modification of H as described above and shown on the left side of Figure 3, it follows that the two edge-disjoint paths problem for directed graphs is also NP-complete. Let H' be the graph H with 6 additional vertices s, t, u_1, u_2, v_1, v_2 , 6 additional undirected edges

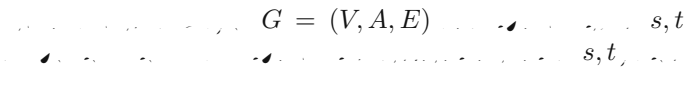
$$\{s, u_1\}, \{u_1, u_2\}, \{u_2, t\}, \{s, v_1\}, \{v_1, v_2\}, \{v_2, t\},$$

and 4 additional directed edges

$$(t_1, u_1), (u_2, s_1), (t_2, v_1), (v_2, s_2),$$

as shown in Figure 3 on the right side. Then every s, t -path in mixed graph H' passes either edge $\{u_1, u_2\}$ or edge $\{v_1, v_2\}$, or both of these edges. If a path uses both of these two edges then there is no second s, t -path which is edge-disjoint with the first one. It is now easy to see that there are two edge-disjoint oriented s, t -paths in H' if and only if there are two edge-disjoint paths in directed graph H , one from s_1 to t_1 and one from s_2 to t_2 .

Note that the two edge-disjoint oriented s, t -paths in H' , if they exist, are not edge-simple, because edge $\{u_1, u_2\}$ and $\{v_1, v_2\}$ are used two times. The NP-completeness of finding two vertex-disjoint oriented s, t -paths follows from Theorem 2 and the construction of the 2-line graph, see Definition 1.

Corollary 1.  $G = (V, A, E)$ \rightarrow $G' = (V', A', E')$ s, t G s_2, t_1 G'

Let us next consider edge-simple mutually edge-disjoint oriented paths. Even, Itai, and Shamir have shown in [EIS76][Theorem 4] that the simple undirected

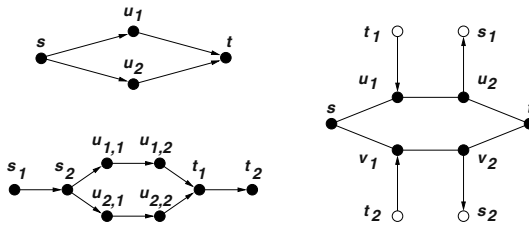


Fig. 3. The left side shows the splitting of the vertices to modify G' as described in the paragraph before Theorem 2. The right side shows the additional vertices and edges inserted into H to get H' in the proof of Theorem 2. Only the filled vertices are added

two commodity integral flow problem is NP-complete. In particular, they have shown the following result. Given an undirected graph $G = (V, E)$, 4 vertices $s_1, t_1, s_2, t_2 \in V$, and two integers $l, k, 1 \leq l \leq k \leq |E|$. The problem to decide whether there are k paths between s_1 and t_1 and l paths between s_2 and t_2 , all mutually edge disjoint, is NP-complete. We use this problem to prove that finding k edge-simple mutually edge-disjoint oriented s, t -paths is also NP-complete.

Theorem 3. Let $G = (V, A, E)$ be an undirected graph with 4 vertices $s_1, t_1, s_2, t_2 \in V$, and two integers $l, k, 1 \leq l \leq k \leq |A|$. Then there are k edge-simple mutually edge-disjoint oriented s_1, t_2 -paths in G if and only if there are $k + l$ mutually edge-disjoint paths in G between s_1, t_1 and s_2, t_2 .

The problem obviously belongs to NP. Let $H = (V, E)$ be an undirected graph, $s_1, t_1, s_2, t_2 \in V$ be 4 vertices, and $l, k, 1 \leq l \leq k \leq |E|$, be two integers. Let H' be the undirected graph H with k additional vertices u_1, \dots, u_k , k additional directed edges, $(t_1, u_1), \dots, (t_1, u_k)$, l additional directed edges, $(u_1, s_2), \dots, (u_l, s_2)$, and $k - l$ additional directed edges, $(u_{l+1}, t_2), \dots, (u_k, t_2)$, see also Figure 4. Then there are k edge-simple mutually edge-disjoint oriented s_1, t_2 -paths in H' if and only if there are $k + l$ mutually edge-disjoint paths in undirected graph H , k paths between s_1 and t_1 and l path between s_2 and t_2 . Since this problem is NP-complete, we have proved the theorem.

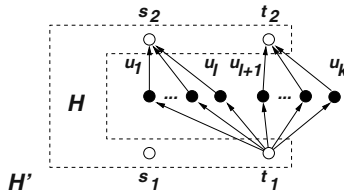


Fig. 4. The additional vertices and edges inserted into H to get H' in the proof of Theorem 3. Only the filled vertices are added

The NP-completeness of finding k vertex-simple mutually vertex-disjoint oriented s, t -paths follows from Theorem 3 and the construction of the k -line graph, see Definition 1.

Corollary 2. Let $G = (V, A, E)$ be a mixed graph and $s, t \in V$ be two vertices of G . Let k be a positive integer, $1 \leq k \leq |A|$. Then there exist k edge-disjoint s, t -paths in G if and only if there exist k vertex-simple edge-disjoint s, t -paths in G .

Next we consider vertex-simple edge-disjoint and edge-simple vertex-disjoint s, t -paths in mixed graphs.

Theorem 4. Let $G = (V, A, E)$ be a mixed graph and $s, t \in V$ be two vertices of G . Then there exist two vertex-simple edge-disjoint s, t -paths in G if and only if there exist two edge-disjoint paths in directed graph H from s_1 to t_1 and from s_2 to t_2 .

The problem obviously belongs to NP. We show a reduction from the two edge-disjoint paths problem for directed graphs. Let $H = (V, A)$ be a directed graph and $s_1, t_1, s_2, t_2 \in V$ be 4 distinct vertices of H . Let H' now be the graph H with 4 additional vertices s, t, u, v , 4 additional undirected edges

$$\{s, u\}, \{u, t\}, \{s, v\}, \{v, t\},$$

and 4 additional directed edges

$$(u, s_1), (t_2, u), (v, s_2), (t_1, v)$$

as shown in Figure 5 on the left side. Then every vertex-simple oriented path from s to t in H' uses either the edges

$$\{s, u\}, (u, s_1), (t_1, v), \{v, t\}$$

or the edges

$$\{s, v\}, (v, s_2), (t_2, u), \{u, t\}.$$

There are two vertex-simple edge-disjoint oriented paths from s to t in H' if and only if there are two edge-disjoint paths in directed graph H , one from s_1 to t_1 and one from s_2 to t_2 . Since the transformation can be done in polynomial time, we have proved the theorem.

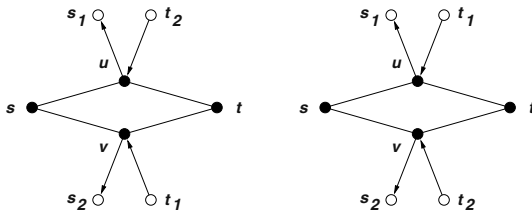


Fig. 5. The left side shows the additional vertices and edges inserted into H to get H' in the proof of Theorem 4. Only the filled vertices are added

The NP-completeness of finding two edge-simple vertex-disjoint oriented s, t -paths can be shown analogously by a reduction from the two vertex-disjoint paths problem for directed graphs. The difference in the construction of H' is shown on the right side of Figure 5.

Corollary 3. Let $G = (V, A, E)$ be a mixed graph and $s, t \in V$ two vertices. Let $C_{s,t} \subseteq A \cup E$ be an s, t -edge cut of G .

Note that the two s, t -paths in mixed graph H' of the proof of Theorem 4 are not vertex-disjoint, and the two s, t -paths in mixed graph H' constructed for Corollary 3 are not vertex-simple.

5 Finding Minimal s, t -Cut Sets

For a given mixed graph $G = (V, A, E)$ and two vertices $s, t \in V$, let $C_{s,t} \subseteq A \cup E$ be a set of edges such that G without the edges of $C_{s,t}$ has no oriented path from s to t . We call such an edge set an s, t -edge cut. An s, t -edge cut is called minimal if there is no s, t -edge cut with fewer edges. Our aim is to compute the size of a minimal s, t -edge cut. Let $c_{s,t}(G)$ be the size of a minimal s, t -edge cut, let $\text{vd-}o_{s,t}(G)$, $\text{vs-}o_{s,t}(G)$, and $\text{es-}o_{s,t}(G)$ be the maximal number of general, vertex-simple, and edge-simple mutually vertex-disjoint oriented s, t -paths, respectively, and let $\text{ed-}o_{s,t}(G)$, $\text{vs-ed-}o_{s,t}(G)$, and $\text{es-ed-}o_{s,t}(G)$ be the maximal number of general, vertex-simple, and edge-simple mutually edge-disjoint oriented s, t -paths, respectively. The following inequalities are easy to verify.

$$\begin{aligned} \text{vd-}o_{s,t}(G) &\leq \text{ed-}o_{s,t}(G) \\ \text{es-}o_{s,t}(G) &\leq \text{es-ed-}o_{s,t}(G) \\ \text{vs-}o_{s,t}(G) &\leq \text{vs-ed-}o_{s,t}(G) \end{aligned}$$

The number of mutually edge-disjoint oriented s, t -paths is always a lower bound on the size of a minimal s, t -edge cut $C_{s,t}$, i.e., $\text{ed-}o_{s,t}(G) \leq c_{s,t}(G)$, because every oriented s, t -path has to contain at least one edge of $C_{s,t}$. The graph of Figure 6 shows an example where $\text{vd-}o_{s,t}(G) = \text{vs-ed-}o_{s,t}(G) = 1$ but $c_{s,t}(G) = 6$. Following this idea of the construction of Figure 6, it is easy to define for every integer k a graph G such that $\text{vd-}o_{s,t}(G) = \text{vs-ed-}o_{s,t}(G) = 1$ and $c_{s,t}(G) = k$.

For every fixed integer k , the problem to decide whether there is an s, t -edge cut $C_{s,t}$ of size at most k can be solved in polynomial time by selecting every subset of $E \cup A$ of size at most k and testing whether the graph without these edges has an oriented path from s to t . This can be done in polynomial time for general, edge-simple, and vertex-simple oriented paths.

However, finding a minimal s, t -edge cut $C_{s,t}$ is NP-complete as the next theorem shows.

Theorem 5. Let $G = (V, A, E)$ be a mixed graph and $s, t \in V$ two vertices. For every integer $k, 1 \leq k \leq |A|$, it is NP-complete to decide whether there is an s, t -edge cut $C_{s,t}$ of G with $|C_{s,t}| \leq k$.

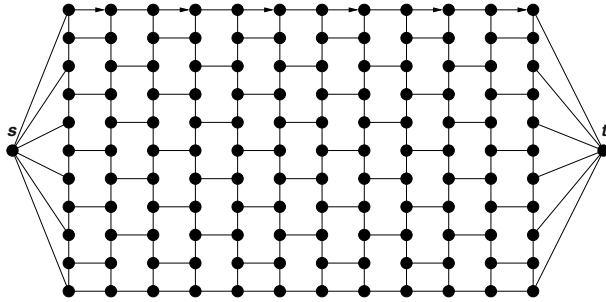


Fig. 6. A mixed graph G where $vd\text{-}o_{s,t}(G) = vs\text{-}ed\text{-}o_{s,t}(G) = 1$, $es\text{-}ed\text{-}o_{s,t}(G) = 2$, $ed\text{-}o_{s,t}(G) = 3$, and $c_{s,t}(G) = 6$

The problem obviously belongs to NP. Let $H = (V, E)$ be an undirected graph and u, v, w be three vertices of H . The problem to decide whether undirected graph H has k edges $e_1, \dots, e_k \in E$ such that H without the edges e_1, \dots, e_k has no path between two distinct vertices of $\{u, v, w\}$ is NP-complete, see [DJP⁺94]. This problem is called the *edge cut problem*. Let $m = |E|$ be the number of edges of H . We now extend the given undirected graph H by $4 \cdot (m + 1) + 2$ additional vertices

$$x_{s,u,1}, \dots, x_{s,u,m+1}, x_{v,t,1}, \dots, x_{v,t,m+1}, x_{s,w,1}, \dots, x_{s,w,m+1}, x_{w,t,1}, \dots, x_{w,t,m+1},$$

s , and t . Then we add $4 \cdot (m + 1)$ directed edges

$$\begin{aligned} & (s, x_{s,u,1}), \dots, (s, x_{s,u,m+1}), (x_{s,u,1}, u), \dots, (x_{s,u,m+1}, u), \\ & (v, x_{v,t,1}), \dots, (v, x_{v,t,m+1}), (x_{v,t,1}, t), \dots, (x_{v,t,m+1}, t) \end{aligned}$$

and $4 \cdot (m + 1)$ undirected edges

$$\begin{aligned} & \{s, x_{s,w,1}\}, \dots, \{s, x_{s,w,m+1}\}, \{x_{s,w,1}, w\}, \dots, \{x_{s,w,m+1}, w\}, \\ & \{w, x_{w,t,1}\}, \dots, \{w, x_{w,t,m+1}\}, \{x_{w,t,1}, t\}, \dots, \{x_{w,t,m+1}, t\}. \end{aligned}$$

Figure 7 shows an example of this construction.

The new mixed graph H' has an oriented path from s to t if and only if H has an undirected path between two distinct vertices of $\{u, v, w\}$. It is not possible to remove all directed paths from s to u or from v to t or all the undirected paths between s and w or w and t by removing at most m edges of H' . Thus, there is an s, t -cut set of size $k \leq m$ if and only if there are k edges in H such that H without these k edges has no path between two distinct vertices of $\{u, v, w\}$. The construction can obviously be done in polynomial time.

Let us finally consider vertex cuts. For a given mixed graph $G = (V, A, E)$ and two vertices $s, t \in V$, let $\bar{C}_{s,t} \subseteq V - \{s, t\}$ be a set of vertices such that G without the vertices of $\bar{C}_{s,t}$ (and without their incident edges) has no oriented path from s to t . We call such a vertex set an s, t -vertex cut. An s, t -vertex cut is called *minimal* if there is no s, t -vertex cut with fewer vertices. Let $\bar{c}_{s,t}(G)$ be the

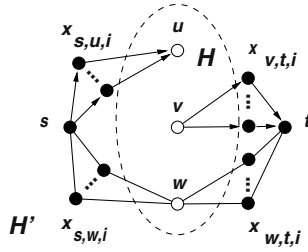


Fig. 7. Graph H' constructed from H as in the proof of Theorem 5

size if a minimal s, t -vertex cut for G , then $\bar{c}_{s,t}(G) \leq c_{s,t}(G)$. The gap between $\text{vd-o}_{s,t}(G)$ ($\text{vs-ed-o}_{s,t}(G)$, respectively) and $\bar{c}_{s,t}(G)$ can also be arbitrary large, see Figure 6 where $\bar{c}_{s,t}(G) = 6$.

The next corollary shows that computing the size of a minimal s, t -vertex cut is also NP-complete.

Corollary 4. Let $G = (V, A, E)$ be an undirected graph with $s, t \in V$, $|V| \geq 3$, $k, 1 \leq k \leq |V| - 2$ and $\bar{C}_{s,t}(G) \geq k$.

The problem obviously belongs to NP. Let $G_{s,t}$ be the graph G with two additional vertices s', t' and two additional undirected edges $\{s', s\}$ and $\{t', t\}$. Let d be the maximal vertex degree of the vertices of G , and let $G_{s,t}^d$ be the d -line graph of $G_{s,t}$, see Definition 1 and the discussion at the end of Section 2.

Since all vertices (u, i) , $u \in V$, $1 \leq i \leq d$, of $G_{s,t}^d$ have the same neighbors, we can assume, without loss of generality, that a minimal $\{s', s\}, \{t', t\}$ -vertex cut $\bar{C}_{\{s',s\},\{t',t\}}$ of $G_{s,t}^d$ consists only of vertices from $A \cup E$. Since these vertices correspond to edges of G , $\bar{C}_{\{s',s\},\{t',t\}}$ defines also an s, t -edge cut of G , and vice versa. Thus $c_{s,t}(G) = \bar{c}_{\{s',s\},\{t',t\}}(G_{s,t}^d)$ and the corollary follows from Theorem 5.

Theorem 5 and Corollary 4 also hold for edge cuts and vertex cuts defined only for vertex-simple or edge-simple oriented paths. Let G be an undirected graph and u, v, w be three vertices of G , let H be the mixed graph with two vertices s, t defined by G and u, v, w as in the proof Theorem 5, and let $H_{s,t}^d$ be d -line graph defined by H and s, t as in the proof of Corollary 4. If G has an undirected path between two vertices of $\{u, v, w\}$ then H has a vertex-simple oriented s, t -path and $H_{s,t}^d$ has a vertex simple oriented $\{s', s\}, \{t', t\}$ -path. Conversely, if $H_{s,t}^d$ has an oriented $\{s', s\}, \{t', t\}$ -path then H has an oriented s, t -path and G has an undirected path between two vertices of $\{u, v, w\}$. This implies by the first observation that $H_{s,t}^d$ has also a vertex-simple oriented $\{s', s\}, \{t', t\}$ -path.

6 Concluding Remarks, Approximations, Acknowledgments

In Section 4, we have shown the NP-completeness of the k oriented mutually disjoint paths problem for the following cases.

	general	edge-simple	vertex-simple
edge-disjoint	case 1: $k \geq 2$	case 3: k unbounded	case 5: $k \geq 2$
vertex-disjoint	case 2: $k \geq 2$	case 4: $k \geq 2$	case 6: k unbounded

The complexity of the k edge-simple mutually edge-disjoint and k vertex-simple mutually vertex-disjoint oriented s, t -paths problems for fixed k is still open.

The proofs of the NP-completeness for case 1, 2, 4, and 5 imply that there are no polynomial time $(2 - \epsilon)$ -approximation algorithms for the corresponding problems. The proofs of the NP-completeness for case 3 and 6 are done by a reduction from the undirected 3-ways edge cut problem which is MAX-SNP-hard. Since the proofs are approximation preserving reductions, there are no polynomial time approximation schemes for the corresponding problems, unless $P=NP$, see [DJP⁺94].

In Section 5, Theorem 5 and Corollary 4, we have shown that computing minimal s, t -edge and s, t -vertex cuts for mixed graphs are NP-complete. However, an s, t -edge cut for G of size at most $2 \cdot c_{s,t}(G)$ can be computed in polynomial time by the following idea. Consider again the two level graph G' of G as defined in Definition 2. If $C_{(s,1),(t,2)}$ is an $(s, 1), (t, 2)$ -edge cut for G' then

$$\begin{aligned} & \{(u, v) \mid ((u, 1), (v, 2)) \in C_{(s,1),(t,2)} \vee ((u, 2), (v, 2)) \in C_{(s,1),(t,2)}\} \\ \cup & \{\{u, v\} \mid \{(u, 1), (v, 1)\} \in C_{(s,1),(t,2)} \vee \{(u, 2), (v, 2)\} \in C_{(s,1),(t,2)}\} \end{aligned}$$

is an s, t -edge cut for G , and conversely, if $C_{s,t}$ is an s, t -edge cut for G then

$$\begin{aligned} & \{((u, 1), (v, 2)) \mid (u, v) \in C_{s,t}\} \\ \cup & \{((u, 2), (v, 2)) \mid (u, v) \in C_{s,t}\} \\ \cup & \{\{(u, 1), (v, 1)\} \mid \{u, v\} \in C_{s,t}\} \\ \cup & \{\{(u, 2), (v, 2)\} \mid \{u, v\} \in C_{s,t}\} \end{aligned}$$

is an $(s, 1), (t, 2)$ -edge cut for G' . Thus, the size of a minimal $(s, 1), (t, 2)$ -edge cut is at most two times the size of a minimal s, t -edge cut for G . Since all $(s, 1), (t, 2)$ -paths in G' are oriented, a minimal $(s, 1), (t, 2)$ -edge cut for G' is computable in polynomial time by network flow techniques, see, for example, [AMO93]. The same idea leads to a polynomial time 2-approximation algorithm for computing a minimal s, t -vertex cut.

We have also shown in Section 5 that the gap between $vd\text{-}o_{s,t}(G)$, $vs\text{-}ed\text{-}o_{s,t}(G)$ and $c_{s,t}(G)$, $\bar{c}_{s,t}(G)$ can be arbitrary large. For edge-simple and general mutually edge-disjoint oriented paths we conjecture the following bounds.

$$\frac{c_{s,t}(G)}{es\text{-}ed\text{-}o_{s,t}(G)} \leq 3, \quad \frac{\bar{c}_{s,t}(G)}{es\text{-}ed\text{-}o_{s,t}(G)} \leq 3, \quad \frac{c_{s,t}(G)}{ed\text{-}o_{s,t}(G)} \leq 2, \quad \text{and} \quad \frac{\bar{c}_{s,t}(G)}{ed\text{-}o_{s,t}(G)} \leq 2.$$

As stated in the introduction, this work is motivated by tracing paths in a graph whose vertices represent 3-dimensional areas of the macaque brain. Undirected edges represent identities and directed edges represent strict inclusions. An area B is properly contained in an area A if and only if there is an oriented path from A to B . Another interesting problem, which we analyze in a forthcoming paper, is to decide whether two areas definitely overlap. This is related to the problem of finding an alternating sequence $u_0, e_1, u_1, \dots, e_k, u_k$ of vertices $u_0, \dots, u_k \in V$ and edges $e_1, \dots, e_k \in A \cup E$ such that for some i , $0 < i < k$, the two sequences $u_0, e_1, u_1, \dots, e_i, u_i$ and $u_k, e_k, u_{k-1}, \dots, e_{i+1}, u_i$ are oriented paths. Such a sequence is called a *forth-and-back path*. A forth-and-back path from s to t , not necessarily simple, can be found in linear time, by finding a mixed path from $(s, 1)$ to $(t, 3)$ in the *three level graph* defined as follows.

Definition 3. Let $G = (V, A, E)$ be a directed graph. The *three level graph* $G' = (V', A', E')$ is defined as follows:

$$\begin{aligned}
 A' = & \{((u, 1), (v, 2)) \mid (u, v) \in A\} \cup \{((u, 2), (v, 2)) \mid (u, v) \in A\} \\
 & \cup \{((v, 2), (u, 3)) \mid (u, v) \in A\} \cup \{((v, 3), (u, 3)) \mid (u, v) \in A\} \\
 E' = & \{\{(u, 1), (v, 1)\} \mid \{u, v\} \in E\} \cup \{\{(u, 2), (v, 2)\} \mid \{u, v\} \in E\} \\
 & \cup \{\{(u, 3), (v, 3)\} \mid \{u, v\} \in E\}
 \end{aligned}$$

Here again, the corresponding s, t -forth-and-back path in G do not need to be simple if the mixed path from $(s, 1)$ to $(t, 3)$ in the three level graph is simple. A forth-and-back path in a mixed graph is a generalization of a so-called *valley-free path* in a directed graph, as considered by Erlebach, Hall, Panconesi, and Vukadinović in [EHPV03] and [Hal03]. A valley-free path in a directed graph $G = (V, A)$ is an alternating sequence $u_0, e_1, u_1, \dots, e_k, u_k$ of vertices $u_0, \dots, u_k \in V$ and edges $e_1, \dots, e_k \in A$ such that for some i , $0 \leq i \leq k$, the two sequences $u_0, e_1, u_1, \dots, e_i, u_i$ and $u_k, e_k, u_{k-1}, \dots, e_{i+1}, u_i$ are directed paths.

We would like to thank Thomas Erlebach for very fruitful discussions at the Internationales Begegnungs- und Forschungszentrum für Informatik, Schloß Dagstuhl, Germany.

References

[AMO93] A. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Prentice Hall, Englewood Cliffs, N.J., 1993.

[DJP⁺94] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. The complexity of multiway cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[EHPV03] T. Erlebach, A. Hall, A. Panconesi, and D. Vukadinović. Cuts and Disjoint Paths in the Valley-Free Path Model. Technical Report 180, Swiss Federal Institute of Technology Zurich, Swiss, 2003.

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

- [Ess85] D.C. Van Essen. Functional organization of primate visual cortex. In E.G. Jones and A. Peters, editors, *Cerebral cortex. III. Visual cortex*, pages 259–329, New York and London, 1985. Plenum Press.
- [FHW80] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.
- [Hal03] A. Hall. Scheduling and Flow-Related Problems in Networks. Dissertation, Swiss Federal Institute of Technology Zurich, Swiss, 2003.
- [Shi80] Y. Shiloach. A polynomial solution to the undirected two paths problem. *Journal of the ACM*, 27:445–456, 1980.
- [SZK00] K.E. Stepfan, K. Zilles, and R. Kötter. Coordinate-independent mapping of structural and functional data by objective relational transformation (ort). *Philosophical Transactions of the Royal Society London, Biological Sciences*, 355:37–54, 2000.
- [Zil04] K. Zilles. Architecture of the Human Cerebral Cortex. Regional and Laminar Organization. In G. Paxinos and J.K. Mai, editors, *The Human Nervous System*, pages 997–1055, San Diego, CA, 2004. Elsevier. 2nd edition.

Polynomial Deterministic Rendezvous in Arbitrary Graphs

Dariusz R. Kowalski^{1,2} and Andrzej Pelc^{3,*}

¹ Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
darek@mpi-sb.mpg.de

² Instytut Informatyki, Uniwersytet Warszawski,
Banacha 2, 02-097 Warszawa, Poland

³ Département d'informatique, Université du Québec en Outaouais,
Hull, Québec J8X 3X7, Canada
Andrzej.Pelc@uqo.ca

Abstract. The rendezvous problem in graphs has been extensively studied in the literature, mainly using a randomized approach. Two mobile agents have to meet at some node of a connected graph. We study deterministic algorithms for this problem, assuming that agents have distinct identifiers and are located in nodes of an unknown anonymous connected graph. Startup times of the agents are arbitrarily decided by the adversary. The measure of performance of a rendezvous algorithm is its *cost*: for a given initial location of agents in a graph, this is the number of steps since the startup of the later agent until rendezvous is achieved. Deterministic rendezvous has been previously shown feasible in arbitrary graphs [16] but the proposed algorithm had cost *exponential* in the number n of nodes and in the smaller identifier l , and polynomial in the difference τ between startup times. The following problem was stated in [16]: Does there exist a deterministic rendezvous algorithm with cost polynomial in n , τ and in labels L_1 , L_2 of the agents (or even polynomial in n , τ and $\log L_1$, $\log L_2$)? We give a positive answer to both problems: our main result is a deterministic rendezvous algorithm with cost polynomial in n , τ and $\log l$. We also show a lower bound $\Omega(n^2)$ on the cost of rendezvous in some family of graphs.

1 Introduction

Two mobile agents located in nodes of an undirected connected graph, have to meet at some node of the graph. This task is known in the literature as the rendezvous problem in graphs, and in this paper we study deterministic algorithms to solve it efficiently. If nodes of the graph are labeled then agents can decide to meet at a predetermined node and the rendezvous problem reduces to graph exploration. However, if the graph models an unknown environment, a unique labeling of nodes may not be available, or agents may be unable to recognize node labels. Hence it is important to design rendezvous algorithms for

* Research supported in part by NSERC grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

agents operating in d -regular graphs, i.e., graphs without unique labeling of nodes. Clearly, the agents must be capable of distinguishing ports at a node: otherwise, it may even be impossible to visit all neighbors of a node of degree 3 (after visiting the second neighbor, it is impossible to distinguish the port leading to the first visited neighbor from that leading to the unvisited one). Consequently, agents initially located at two such nodes, might never be able to meet. Hence we make a natural assumption that all ports at a node are locally labeled $1, \dots, d$, where d is the degree of the node. No coherence between local port labelings is assumed. We do not assume any knowledge of the topology of the graph, of its size, or of the distance separating the agents.

1.1 The Model

Synchrony and Startup Times. Agents move in synchronous steps. In every step, an agent may either remain in the same node or move to an adjacent node. We assume that startup times of the agents are arbitrarily decided by an adversary. Agents are not aware of the difference between startup times, and each of them starts executing the rendezvous algorithm and counting steps since its own startup. The agent who starts earlier and happens to visit the starting node of the later agent at the startup of this later agent, is not aware of this fact, i.e., we assume that agents are created at their startup time rather than waiting in the node before it.

Adversarial Decisions and Cost of Rendezvous. An agent, currently located at a node, is not aware of the other endpoints of yet unexplored incident edges. If the agent decides to traverse such a new edge, the choice of the actual edge belongs to the adversary, as we are interested in the worst-case performance. If agents get to the same node in the same round, they become aware of it and rendezvous is achieved. The cost of a rendezvous algorithm, for a given initial location of agents in a graph, is the worst-case number of steps since the startup of the later agent until rendezvous is achieved, where the worst case is taken over all adversary decisions, whenever an agent decides to explore a new edge adjacent to a currently visited node, and over all possible startup times. In particular, time of local computations performed by agents does not contribute to cost.

Labels and Local Knowledge. If agents are identical, i.e., they do not have distinct identifiers, and execute the same algorithm, then deterministic rendezvous is impossible even in the simplest case of simultaneously starting agents in a two-node graph. Hence we assume that agents have distinct labels, which are two different integers, and that every agent knows its own label. (For technical reasons we assume that labels are larger than 1. This assumption can be easily omitted.) If agents knew each other's labels, the problem could be again reduced to that of graph exploration: the agent with smaller label does not move, and the other agent searches the graph until it finds it. (This strategy is sometimes called “wait for mommy”.) However, the assumption that agents know each other may often be unrealistic, as they may be created in different parts of the network in a distributed fashion, oblivious of each other. Hence we assume that each agent knows its own label but does not know the label of the other. The only

initial input of a (deterministic) rendezvous algorithm executed by an agent is the agent's label. During the execution of the algorithm, an agent learns the local port number by which it enters a node and the degree of the node.

Notation. Labels of agents are denoted by L_1 and L_2 . The agent with label L_i is called agent i . (An agent does not know its number, only the value of its label). Labels are distinct integers larger than 1. l denotes the smaller of the two labels. The difference between startup times of the agents is denoted by τ . We use the word "graph" to mean a simple undirected connected graph with local port labelings but without node labels. n denotes the number of nodes in the graph.

1.2 Our Results

In [16], deterministic rendezvous was considered under the above described model. The authors formulated the following two questions:

- Q1.** Is rendezvous feasible in arbitrary graphs?
- Q2.** If so, can it be performed in cost polynomial in n , τ , L_1 and L_2 (or even polynomial in n , τ , $\log L_1$ and $\log L_2$)?

They gave an affirmative answer to the first question but their rendezvous algorithm had cost exponential in n and l (and polynomial in τ). The second question was left open.

We give a positive answer to both versions of this question. Our main result is a deterministic rendezvous algorithm with cost polynomial in n , τ and $\log l$. The algorithm contains a non-constructive ingredient: agents use combinatorial objects whose existence we prove by the probabilistic method. Nevertheless our algorithm is indeed deterministic. Both agents can find separately the same combinatorial object with desired properties (which is then used in the rendezvous algorithm). This can be done using brute force exhaustive search which may be quite complex but in our model only moves of the agents are counted and computation time of the agents does not contribute to cost. Moreover, it should be noticed that finding this combinatorial object can be done a single time at a preprocessing stage, the object can be stored in agents' memory and subsequently used in many instances of the rendezvous problem. We also show a lower bound $\Omega(n^2)$ on rendezvous cost in some family of graphs.

The paper is organized as follows. In Section 2 we construct a simpler rendezvous algorithm polynomial in n , τ and l (instead of $\log l$). We do this to first present the main idea of the algorithm and of its analysis without additional complications needed to decrease the cost. In Section 3 we show how to modify this algorithm, in order to decrease its cost to polynomial in n , τ and $\log l$. In Section 4 we establish the lower bound $\Omega(n^2)$ on rendezvous cost in some family of graphs. Section 5 contains conclusions and open problems.

1.3 Related Work

The rendezvous problem has been introduced in [23]. The vast body of results on rendezvous (see the book [4] for a complete discussion and more ref-

erences) can be divided into two classes: papers considering the geometric scenario (rendezvous in the line, see, e.g., [11, 12, 19], or in the plane, see, e.g., [9, 10]), and those discussing rendezvous in graphs, e.g., [2, 5]. Most of the papers, e.g., [2, 3, 7, 11, 20] consider the probabilistic scenario: inputs and/or rendezvous strategies are random. In [20] randomized rendezvous strategies are applied to study self-stabilized token management schemes. Randomized rendezvous strategies use random walks in graphs, which have been widely studied and applied also, e.g., in graph traversing [1], on-line algorithms [14] and estimating volumes of convex bodies [17]. A natural extension of the rendezvous problem is that of gathering [18, 20, 22, 24], when more than 2 agents have to meet in one location.

Deterministic rendezvous with anonymous agents working in unlabeled graphs but equipped with tokens used to mark nodes was considered e.g., in [21]. In [25] the authors considered rendezvous of many agents with unique labels. Although one of their scenarios is deterministic, it differs from our setting in that agents know the graph and they know a finite set containing the team of agents that are supposed to meet. Deterministic rendezvous in unlabeled graphs, assuming that each agent knows only its own identity, was considered in [16]. The authors considered rendezvous under the scenario adopted in the present paper, and under another scenario which additionally assumed simultaneous startup. They gave efficient rendezvous algorithms for trees and rings and proved feasibility of rendezvous for arbitrary graphs. In the case of arbitrary startup times (which we assume in the present paper) their algorithm for arbitrary graphs was $O(n^2 \log n)$ in n and l (and polynomial in τ).

2 A Rendezvous Algorithm Polynomial in n , τ and l

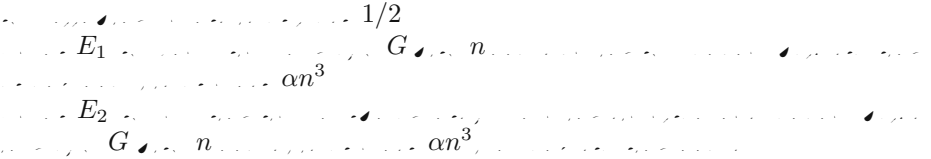
2.1 Deterministic Polynomial Covering of a Graph

A k -path of length k in a graph is a sequence (v_1, \dots, v_k) of nodes such that node v_{i+1} is adjacent to v_i , for all $i < k$. A k -cover is a walk in which every node of the graph appears at least once. The aim of this subsection is to give a deterministic procedure, using a number of steps polynomial in n which, when started in any node of an unknown graph with at most n nodes, produces a covering walk in this graph. This procedure will be an important ingredient in our rendezvous algorithms.

Define a d -step of an agent in graph G as a walk in which the agent, currently located at a node of degree d , acts in the next step as follows: it remains in the node with probability $1/2$ and moves through any port with the same probability $1/(2d)$. The d -cover of the graph G during a random walk starting at node v is the random variable denoting the smallest number of steps after which the agent performing this walk visits all nodes of the graph. The d -meet of two agents performing simultaneous random walks in graph G , starting at nodes v and w , is the random variable denoting the smallest number of steps after which agents performing these walks meet at some node.

We will use the following Lemma proved in [15]:

Lemma 1. For any $\alpha > 0$



Let α be the constant from Lemma 1. Let $\lambda(n) = \lceil 2\alpha n^5 \log n \rceil$. The next lemma shows a useful property of a random walk (the proof is omitted).

Lemma 2. For any graph G with n nodes and any node v in G , the probability that a random walk of length $\lambda(n)$ starting at v does not cover all nodes is at most $1 - 2^{-2n^2 \log n}$.

For any positive integer n and any function $h_n : \{1, \dots, \lambda(n)\} \times \{1, \dots, n - 1\} \rightarrow \{0, 1, \dots, n - 1\}$, such that $h_n(i, d) \leq d$, we define the following procedure describing a walk of length $\lambda(n)$ in a graph G , starting at a node v (cf. the upper bound for the length of a universal traversal sequence [1]).

Procedure GRAPHCOVER(n, h_n)

In step $i \leq \lambda(n)$, the agent, currently located at a node of degree d , moves to an adjacent node by port $h_n(i, d)$, or remains idle if $h_n(i, d) = 0$. After step $\lambda(n)$ it stops.

Lemma 3. For any n and any function $h_n : \{1, \dots, \lambda(n)\} \times \{1, \dots, n - 1\} \rightarrow \{0, 1, \dots, n - 1\}$, such that $h_n(i, d) \leq d$, the probability that a random walk of length $\lambda(n)$ in a graph G starting at a node v does not cover all nodes is at most $2^{-2n^2 \log n}$.

Fix n . Fix a graph G with at most n nodes and fix some starting node v in G . We can do it in at most $n^{n^2} \cdot n \leq 2^{n^2(\log n + 1)}$ different ways, for fixed n . Applying Lemma 2, the probability of the event ‘there exists a graph G with at most n nodes and a starting node v in G , such that the random walk of length $\lambda(n)$ in graph G starting in v is not a covering walk’ is at most

$$2^{-2n^2 \log n} \cdot 2^{n^2(\log n + 1)} \leq 2^{-n}.$$

Using the probabilistic argument we prove the existence of the desired function, which completes the proof. \square

Note that the problem of construction of function h_n satisfying Lemma 3 is hard (cf. hardness of a construction of a universal traversal sequence even for 3-regular graphs [13]).

In our applications to rendezvous algorithms, agents will use Procedure GRAPHCOVER(n, h_n) producing a covering walk in any graph with at most n nodes. To this end we want each of the agents to find the same function h_n whose existence is guaranteed by Lemma 3. (This can be done by exhaustive search, ordering all such possible functions in a canonical way and checking them one by one to find the first suitable one. Recall that, according to our model, only moves of the agents are accounted for, and computation time of the agents does not contribute to rendezvous cost.) Let \hat{h}_n be the first function in this canonical ordering, satisfying Lemma 3, for any n . To simplify notation, we will write GRAPHCOVER(n) instead of GRAPHCOVER(n, \hat{h}_n), throughout the paper.

2.2 Construction and Analysis of Rendezvous Algorithm PA

In order to design rendezvous algorithm PA, we will use procedure GRAPHCOVER(n), which takes $\lambda(n) = \lceil 2\alpha n^5 \log n \rceil$ steps and produces a covering walk in any graph with at most n nodes.

We will show that the following algorithm completes rendezvous in any n -node graph, for agents with arbitrary labels L_1, L_2 , with arbitrary delay τ , in cost polynomial in $n, l = \min\{L_1, L_2\}$ and τ .

Algorithm PA (PassiveActive) for agent with label L .

For $k = 1, 2, \dots$ **do**

Passive Phase: Wait for $2Lk$ steps

Active Phase:

- Perform GRAPHCOVER(Lk), starting from the current node in the graph
- Perform L times GRAPHCOVER(k), always starting from the current node in the graph

Let $k_0 = \lambda(n)$. The idea of the algorithm is to guarantee that one of the agents is passive while the other agent L performs GRAPHCOVER(k_0) and thus completes rendezvous. (We refer to this situation by saying that the active agent meets the passive agent – an asymmetric relation.) This is the reason for having increasing time segments of activity and passivity. The turn of the “for” loop for a given k will be called the k th epoch of the agent. The k th epoch of agent with label L has two epochs of equal length $2Lk$: the passive phase and the active phase. The active phase is composed of an execution of GRAPHCOVER(Lk) followed by L executions of GRAPHCOVER(k). This is the subtle point in the algorithm design: it seems that none of these parts alone (one long execution of GRAPHCOVER or many short executions of it) permits to guarantee rendezvous in cost polynomial in n, l and τ .

We analyze the performance of algorithm PA as a function of n, l and τ . Let G be an n -node graph and L_1, L_2 the labels of agents. Without loss of generality assume that $L_1 > L_2 = l$. We start counting time steps from the startup of the later agent. For every step t denote by $k_i(t)$, for $i = 1, 2$, the number of epoch executed by agent i in step t . We will use the following fact, which follows from the Properties of GRAPHCOVER(k), and from the definition of k_0 .

Fact 1 1.

If $k \geq k_0$ and $t \geq 0$, then $L_1 k \geq 2k_0$ implies that the active phase of agent 1 overlaps the active phase of agent 2 in the interval $[t, t + k_0)$.

2. $k \geq k_0$

implies that the active phase of agent 1 overlaps the active phase of agent 2 in the interval $[t, t + 2k_0)$.

3. $k \geq k_0$

implies that the active phase of agent 1 overlaps the active phase of agent 2 in the interval $(t - k_0, t]$.

The following lemma estimates cost of rendezvous under some technical conditions (the proof will appear in the full version of the paper).

Lemma 4. For t_1, t_2 such that $L_1 k_1(t_1) \geq 40k_0$, $k_2(t_2) \geq 10k_0$, $|t_1 - t_2| \leq 4k_0$ and $|L_1 k_1(t_1) - L_2 k_2(t_2)| \leq 2k_0$ it holds that $t_2 + 26lk_2(t_2)k_0$

The next three lemmas estimate the time step by which rendezvous is completed, depending on the number of epoch changes of one agent during one epoch of the other (their proofs will appear in the full version of the paper).

Lemma 5.

1. For t such that $k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$ it holds that $t + 11lk_2(t) + 26lk_2(t)k_0$

2. For t such that $k_2(t) \geq 40k_0$ and $k_1(t) \geq 10k_0$ it holds that $t + 11lk_2(t) + 26lk_2(t)k_0$

Lemma 6.

1. For t such that $k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$ it holds that $t + 14lk_2(t) + 26lk_2(t)k_0$

2. For t such that $k_2(t) \geq 40k_0$ and $k_1(t) \geq 10k_0$ it holds that $t + 14lk_2(t) + 26lk_2(t)k_0$

Lemma 7.

1. For t such that $k_1(t) \geq 40k_0$ and $k_2(t) \geq 10k_0$ it holds that $t + 9lk_2(t)$

2. For t such that $k_2(t) \geq 40k_0$ and $k_1(t) \geq 10k_0$ it holds that $t + 4lk_2(t)$

Theorem 1.

Let G be a graph with n vertices and $L_1 > L_2 = l$. Then the time complexity is $\mathcal{O}(\sqrt{lrn^5 \log n} + ln^{10} \log^2 n)$

Let t_1 be the first step for which $L_1 k_1(t_1) \geq 40k_0$. Let $t_2 \geq t_1$ be the first step for which $k_2(t_2) \geq 10k_0$. Observe that t_i is the beginning of epoch $k_i(t_i)$ of the i th agent. Consider the step $t^* = t_2 + (t_2 - t_1) + 8L_2 k_2(t_2) = 2t_2 + 8L_2 k_2(t_2) - t_1$. We have $t^* > t_2 \geq t_1$, hence $k_1(t_1) \leq k_1(t^*)$. Consider two cases.

Case A. $k_1(t_1) = k_1(t^*)$.

We have the inclusion $[t_2, t_2 + 4L_2k_2(t_2)] \subseteq [t_1, (t_1 + t^*)/2]$. Hence the epoch $k_2(t_2)$ of the second agent is included in the passive phase of epoch $k_1(t_1)$ of the first agent. We use Fact 1 point 1 to obtain that rendezvous is completed during epoch $k_2(t_2)$ of the second agent. Hence rendezvous cost is $\mathcal{O}(t_2 + L_2k_2(t_2))$. By definition of t_1 we get that $t_1 \in \mathcal{O}(k_0^2) = \mathcal{O}(n^{10} \log^2 n)$.

If $t_2 > t_1$ then $k_2(t_1) < 10k_0$, and consequently $k_2(t_2) = 10k_0$. Hence $t_2 = \mathcal{O}(L_2k_0^2) \in \mathcal{O}(ln^{10} \log^2 n)$. On the other hand, $L_2k_2(t_2) \in \mathcal{O}(ln^5 \log n)$.

If $t_2 = t_1$ then $t_2 \in \mathcal{O}(n^{10} \log^2 n)$. On the other hand we have

$$4L_2(k_2(1) + 1) + \dots + 4L_2(k_2(t_2) - 1) \leq t_2.$$

Hence $k_2(t_2) - k_2(1) \in \mathcal{O}(\sqrt{t_2/L_2})$. It follows that $L_2k_2(t_2) \in \mathcal{O}(L_2(k_2(1) + \sqrt{t_2/L_2}))$. Since $\tau \in \Omega(L_2(k_2(1))^2)$, we get that $k_2(1) \in \mathcal{O}(\sqrt{\tau/L_2})$, and hence rendezvous cost is

$$\mathcal{O}(t_2 + L_2k_2(t_2)) \subseteq \mathcal{O}(t_2 + L_2\sqrt{\tau/L_2} + L_2\sqrt{t_2/L_2}) \subseteq \mathcal{O}(n^{10} \log^2 n + \sqrt{l\tau} + \sqrt{ln^5 \log n}).$$

Consequently, in both situations, rendezvous cost is $\mathcal{O}(ln^{10} \log^2 n + \sqrt{l\tau})$.

Case B. $k_1(t_1) < k_1(t^*)$.

In this case we have that $t_2 \in \mathcal{O}(k_0^2 l)$ and $k_2(t_2) \in \mathcal{O}(\sqrt{\tau/l} + k_0)$. Below we give the proof of this statement in all possible situations.

- $L_1k_1(1) \geq 40k_0$ and $k_2(1) \geq 10k_0$. In this case $t_2 = t_1 = 1$. We also have $\tau \in \Omega(L_2(k_2(1))^2)$, and consequently $k_2(t_2) = k_2(1) \in \mathcal{O}(\sqrt{\tau/l})$.
- $L_1k_1(1) \geq 40k_0$ and $k_2(1) < 10k_0$. In this case $t_1 = 1$ and $t_2 \in \mathcal{O}(L_2k_0^2) = \mathcal{O}(lk_0^2)$. Also $k_2(t_2) \in \mathcal{O}(k_0)$.
- $L_1k_1(1) < 40k_0$ and $k_2(1) \geq 10k_0$. In this case $L_1(k_1(t_1) - 1) < 40k_0$, which implies that $t_1 \in \mathcal{O}(L_1(k_1(t_1))^2) \subseteq \mathcal{O}(k_0^2/L_1)$. Also $t_2 = t_1$, which gives $t_2 \in \mathcal{O}(k_0^2)$. On the other hand, $\tau \in \Omega(L_2(k_2(1))^2)$, and consequently $k_2(t_2) \in k_2(1) + \mathcal{O}(k_0) \subseteq \mathcal{O}(\sqrt{\tau/l} + k_0)$.
- $L_1k_1(1) < 40k_0$ and $k_2(1) < 10k_0$. In this case $L_1(k_1(t_1) - 1) < 40k_0$, which implies that $t_1 \in \mathcal{O}(L_1(k_1(t_1))^2) \subseteq \mathcal{O}(k_0^2/L_1)$. Also $t_2 \in t_1 + \mathcal{O}(L_2k_0^2) = \mathcal{O}(lk_0^2)$. On the other hand, $k_2(t_2) \leq k_2(t_1) + 10k_0 \in k_2(1) + \mathcal{O}(k_0) = \mathcal{O}(k_0)$.

Let t be the first step after t_2 in which an epoch of the first agent starts. Notice that, by the assumption $k_1(t_1) < k_1(t^*)$, we have $t \leq t_2 + t^*$, and consequently $t \in \mathcal{O}(t_2 + L_2k_2(t_2))$. Hence $k_2(t) \in \mathcal{O}(k_2(t_2))$. Consider times $t'_1, t'_2 > t$ such that t'_i is the end of epoch $k_i(t)$, for $i = 1, 2$.

Subcase B1. $t'_1 \leq t'_2$.

Consider epoch $k_2(t)$ of the second agent. By definition of step t , this epoch starts not earlier than t_2 . Since $t'_1 \leq t'_2$, we have that the first agent ends its epoch at least once during epoch $k_2(t)$ of the second agent. Applying point 2 of one of the Lemmas 5, 6 and 7, depending on the number of epoch changes of the first agent in epoch $k_2(t_2)$, we obtain that rendezvous cost is at most

$$\begin{aligned} t + 18lk_2(t) + 26lk_2(t)k_0 &\in \mathcal{O}(t_2 + L_2k_2(t_2)) + \mathcal{O}(lk_2(t_2)) + \mathcal{O}(lk_2(t_2)k_0) \subseteq \\ &\subseteq \mathcal{O}(k_0^2 l + l(\sqrt{\tau/l} + k_0) + l(\sqrt{\tau/l} + k_0)k_0) = \mathcal{O}(\sqrt{l\tau}n^5 \log n + ln^{10} \log^2 n). \end{aligned}$$

Subcase B2. $t'_1 > t'_2$.

Consider epoch $k_1(t)$ of the first agent. It starts in step $t \geq t_2$. Since $t'_1 > t'_2$, we have that the second agent ends its epoch at least once during epoch $k_1(t)$ of the first agent. Applying point 1 of one of the Lemmas 5, 6 or 7, depending on the number of epoch changes of the second agent in epoch $k_1(t_1)$, we obtain that rendezvous cost is at most

$$t + 18lk_2(t) + 26lk_2(t)k_0 \in \mathcal{O}(t_2 + L_2k_2(t_2)) + \mathcal{O}(lk_2(t_2)) + \mathcal{O}(lk_2(t_2)k_0) \subseteq \mathcal{O}(lk_0^2 + l(\sqrt{\tau/l} + k_0) + l(\sqrt{\tau/l} + k_0)k_0) \subseteq \mathcal{O}(\sqrt{l\tau}n^5 \log n + ln^{10} \log^2 n),$$

the same asymptotic bound as in Subcase B1. □

3 A Rendezvous Algorithm Polynomial in n, τ and $\log l$

In this section we design and analyze a modification of Algorithm PA which works in cost polynomial in n, τ and $\log l$, rather than polynomial in n, τ and l . The modified algorithm has two non-constructive ingredients: the function determining the covering walk, already used in Procedure GRAPHCOVER(n) and another one, used in the new procedure TRAVERSE described below. (As before, the new combinatorial object (a family of functions) whose existence we prove using again the probabilistic method, can be found by each of the agents separately, using local exhaustive search.) Similarly as before, our algorithm remains deterministic.

Assume that, for every label L and positive integer k , we have a function $f_{L,k} : \{1, \dots, k \lceil \log L \rceil\} \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+ \cup \{0\}$ such that $f_{L,k}(i, d) \leq d$ for any positive integers i, d . We call such a function a *port number function*. The interpretation of $f_{L,k}(i, d)$ is the port number used by agent with label L in the i th step of graph traversal with parameter k , if the agent is currently at a node of degree d (the value 0 indicates that the agent remains at the current node). According to this intuition we define the procedure TRAVERSE. For a non-negative integer t and for a positive integer k , define $T(k, t)$ as the set of all infinite sequences (t_1, \dots, t_k, \dots) of non-negative integers such that $t_1 + \dots + t_k = t$ and $t_i = 0$ for every $i > k$. For a given label L , integers $k > 0$ and $t \geq 0$, fix $\bar{t} \in T(k, t)$ and define:

Procedure TRAVERSE(L, k, \bar{t})

For $i = 1, 2, \dots, k$ **do** initialize $count_i := t_i$ (t_i is the i th value of \bar{t})

For $j = 1, 2, \dots, k \lceil \log L \rceil$ **do**

Set d to the degree of the current node (if $d > k$ and $count_d$ not initialized then initialize $count_d := 0$)

Set $count_d := count_d + 1$

If $f_{L,k}(count_d, d) > 0$ **then Go** using port $f_{L,k}(count_d, d)$

The intuition behind the parameter \bar{t} in the above procedure is the following. For every d , we suppose that, before starting procedure TRAVERSE, t_d first port choices in nodes of degree d , yielded by the function $f_{L,k}$, were already executed. Hence this introduces a shift of procedure TRAVERSE by t steps back, assuming that t_d nodes of degree d were already visited. In the algorithm we will only use TRAVERSE for parameter $\bar{0}$ (hence no shift at all) but in the analysis we will consider executions of TRAVERSE shifted in time with respect to each other, and hence this more general formulation of the procedure will become useful.

The following algorithm uses procedure TRAVERSE, which in turn depends on a family of port-functions. We will use the algorithm for such a family of functions with a specially defined property.

Algorithm MPA (Modified PassiveActive) for agent with label L .

For $k = 1, 2, \dots$ **do**

Passive Phase: Wait for $2 \cdot 2^{\lceil \log L \rceil} k$ steps

Active Phase:

First Stage: Perform GRAPHCOVER($\lceil \log L \rceil k$), starting from the current node in the graph

Middle Stage: Perform TRAVERSE($L, k, \bar{0}$)

Last Stage: Perform $2^{\lceil \log L \rceil}$ times GRAPHCOVER(k), always starting from the current node in the graph

3.1 Choosing Port-Functions

As before, the turn of the “for” loop for a given k will be called the k th epoch of the agent. Let G be an n -node graph, v_1, v_2 two nodes in G , t a non-negative integer, $ind \in \{1, 2\}$, $\bar{t} \in T(k_1, t)$, if $ind = 1$ and $\bar{t} \in T(k_2, t)$ otherwise. Execute($L_1, k_1, L_2, k_2, G, v_1, v_2, \bar{t}, ind$), denotes the execution of procedures

- TRAVERSE(L_1, k_1, \bar{t}) and TRAVERSE($L_2, k_2, \bar{0}$) if $ind = 1$
- TRAVERSE($L_1, k_1, \bar{0}$) and TRAVERSE(L_2, k_2, \bar{t}) if $ind = 2$

by agents operating in graph G , where procedure TRAVERSE(L_1, k_1, \cdot) starts in node v_1 and procedure TRAVERSE(L_2, k_2, \cdot) starts in node v_2 . We assume that procedure Execute is performed until one of the agents completes its procedure TRAVERSE.

Let $\alpha > 0$ be the constant from Lemma 1 and let $k_0 = \lceil 2\alpha n^5 \log n \rceil$ be as in Section 2. We say that a family of port-functions $\{f_{L,k} : k \in \mathbb{Z}^+, L = 2, 3, \dots\}$ is a *rendezvous family*, if the following property is satisfied:

RV for all labels $L_1 > L_2$ such that $\lceil \log L_1 \rceil = \lceil \log L_2 \rceil$, all parameters $k_1 = k_2$, for every n such that $10\alpha n^5 \log n \leq k_1$, every n -node graph G , for all starting nodes v_1, v_2 , any sequence $\bar{t} \in T(k_1, t)$, where $t < 6k_0$, any $ind \in \{1, 2\}$, agents with labels L_1, L_2 meet during procedure Execute($L_1, k_1, L_2, k_2, G, v_1, v_2, \bar{t}, ind$).

Recall that, in view of Lemma 3 and of the choice of α , procedure GRAPHCOVER(n), lasting k_0 steps, produces a covering walk in any graph G with n nodes.

Our goal is to show the existence of a rendezvous family. The proof uses the probabilistic method and requires the analysis of simultaneous random walks of two agents in a graph. The proofs of the next two lemmas will appear in the full version of the paper.

Lemma 8. *Let $L_1 > L_2$ such that $\lceil \log L_1 \rceil = \lceil \log L_2 \rceil$, all parameters $k_1 = k_2$, for every n such that $10\alpha n^5 \log n \log L_1 \leq k_1$, every n -node graph G , for all starting nodes v_1, v_2 , any sequence $\bar{t} \in T(k_1, t)$, where $t < 6k_0$, any $ind \in \{1, 2\}$, agents with labels L_1, L_2 meet during procedure Execute($L_1, k_1, L_2, k_2, G, v_1, v_2, \bar{t}, ind$) with probability at least $1 - 2^{-10n^2 \log n \log L_1}$.*

Lemma 9. *Let $\alpha > 0$ be the constant from Lemma 1 and let $k_0 = \lceil 2\alpha n^5 \log n \rceil$. Let $\{f_{L,k} : k \in \mathbb{Z}^+, L = 2, 3, \dots\}$ be a family of port-functions satisfying the property RV.*

3.2 Analysis of Algorithm MPA

In view of Lemma 9 we can use a rendezvous family of port-functions as a basis for algorithm MPA. The rest of our analysis assumes that MPA uses such a fixed family (which agents can compute locally), and hence we assume that property RV is satisfied. Notice that, in the proof of Lemma 9, we used the probabilistic method for fixed k, L and n . The function $f_{L',k}$ of an agent with label L' , such that $L/2 < L' \leq L$, has the domain bounded by $nk \log L$ and the range bounded by n . The agent may have to compute all such functions $f_{L'',k}$, for $L/2 < L'' \leq L$. This can be done locally in time exponential in $nk \log L \log n$. Recall that local computations do not affect rendezvous cost in our model.

Our next lemma corresponds to Lemma 4 in the analysis of algorithm PA. The proof will appear in the full version of the paper.

Lemma 10. $t_1, t_2, k_1(t_1), k_2(t_2), \lceil \log L_1 \rceil = \lceil \log L_2 \rceil, | \lceil \log L_1 \rceil k_1(t_1) - \lceil \log L_2 \rceil k_2(t_2) | < 2k_0, k_1(t_1), k_2(t_2) \geq 10\alpha n^5 \log n, |t_1 - t_2| < 6k_0$

Theorem 2. $G, L_1 > L_2 = l, \tau, n, \mathcal{O}(n^5 \sqrt{\tau \log l} \log n + n^{10} \log^2 n \log l)$

The proof of Theorem 2 will appear in the full version of the paper.

4 A Lower Bound

The sharpest lower bound for deterministic rendezvous proved in [16] was $\Omega(n \log l)$. More precisely, the fact that rendezvous sometimes requires this cost follows from the lower bound $\Omega(D \log l)$ proved in [16] for agents starting at distance D in a ring. We show that, in some graphs with $\Theta(n^2)$ edges, the cost of rendezvous is $\Omega(n^2)$, i.e., a large part of the graph has to be explored before agents can meet. The proof will appear in the full version of the paper.

Theorem 3. $n, G_n, L_1, L_2, L_1, L_2, \Omega(n^2)$

5 Conclusion

Our main result was the design and analysis of a deterministic rendezvous algorithm polynomial in n, τ and $\log l$. This answers affirmatively question Q2 from [16]. Our algorithm requires exhaustive local search by each agent to find an object whose existence is proved using the probabilistic method. While local computations (even possibly very extensive), do not affect cost in our model, it is interesting to know if there is a deterministic rendezvous algorithm with cost polynomial in $n, \tau, \log l$ whose local computations also take polynomial time.

Another open problem concerns the dependence of rendezvous cost on the parameter τ (the difference between startup times). We showed a lower bound

$\Omega(n^2)$ on rendezvous cost in some graphs. It was shown in [16] that cost $\Omega(\log l)$ is required even for the two-node graph. It was also shown in [16] that, for agents starting at distance $\Omega(n)$ in a ring, cost $\Omega(n \log l)$ is required, even for $\tau = 0$. However, we do not know if any non-constant function of τ is a lower bound on rendezvous cost in some graphs. (Recall that the cost of our algorithm contains a factor $\sqrt{\tau}$.) Hence the following problem remains open:

Does there exist a deterministic rendezvous algorithm whose cost is polynomial in n and l (or even in n and $\log l$) but independent of τ ?

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, Proc. 20th Annual Symposium on Foundations of Computer Science (FOCS'1979), 218-223.
2. S. Alpern, The rendezvous search problem, SIAM J. on Control and Optimization 33 (1995), 673-683.
3. S. Alpern, Rendezvous search on labelled networks, Naval Research Logistics 49 (2002), 256-274.
4. S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, Kluwer Academic Publisher, 2002.
5. J. Alpern, V. Baston, and S. Essegaiar, Rendezvous search on a graph, Journal of Applied Probability 36 (1999), 223-231.
6. S. Alpern and S. Gal, Rendezvous search on the line with distinguishable players, SIAM J. on Control and Optimization 33 (1995), 1270-1276.
7. E. Anderson and R. Weber, The rendezvous problem on discrete locations, Journal of Applied Probability 28 (1990), 839-851.
8. E. Anderson and S. Essegaiar, Rendezvous search on the line with indistinguishable players, SIAM J. on Control and Optimization 33 (1995), 1637-1642.
9. E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, Proc. 14th Annual ACM Symp. on Computational Geometry, 1998.
10. E. Anderson and S. Fekete, Two-dimensional rendezvous search, Operations Research 49 (2001), 107-118.
11. V. Baston and S. Gal, Rendezvous on the line when the players' initial distance is given by an unknown probability distribution, SIAM J. on Control and Optimization 36 (1998), 1880-1889.
12. V. Baston and S. Gal, Rendezvous search when marks are left at the starting points, Naval Res. Log. 48 (2001), 722-731.
13. S.A. Cook and P. McKenzie, Problems complete for deterministic logarithmic space, Journal of Algorithms 8 (5) (1987), 385-394.
14. D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir, Random walks on weighted graphs, and applications to on-line algorithms, Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC'1990), 369-378.
15. D. Coppersmith, P. Tetali, and P. Winkler, Collisions among random walks on a graph, SIAM J. on Discrete Math. 6 (1993), 363-374.
16. A. Dessmark, P. Fraigniaud, and A. Pelc, Deterministic rendezvous in graphs, Proc. 11th European Symposium on Algorithms (ESA'2003), LNCS 2832, 184-195.
17. M. Dyer, A. Frieze, and R. Kannan, A random polynomial time algorithm for estimating volumes of convex bodies, Proc. 21st Annual ACM Symposium on Theory of Computing (STOC'1989), 375-381.

18. P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous oblivious robots with limited visibility, Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2001), LNCS 2010, 247-258.
19. S. Gal, Rendezvous search on the line, Operations Research 47 (1999), 974-976.
20. A. Israeli and M. Jalfon, Token management schemes and random walks yield self stabilizing mutual exclusion, Proc. 9th Annual ACM Symposium on Principles of Distributed Computing (PODC'1990), 119-131.
21. E. Kranakis, D. Krizanc, N. Santoro and C. Sawchuk, Mobile agent rendezvous in a ring, Proc. 23rd International Conference on Distributed Computing Systems (ICDCS'2003), 592-599.
22. W. Lim and S. Alpern, Minimax rendezvous on the line, SIAM J. on Control and Optimization 34 (1996), 1650-1665.
23. T. Schelling, The strategy of conflict, Oxford University Press, Oxford, 1960.
24. L. Thomas, Finding your kids when they are lost, Journal on Operational Res. Soc. 43 (1992), 637-639.
25. X. Yu and M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, Proc. International Colloquium on Automata, Languages, and Programming (ICALP'1996), LNCS 1099, 610-621.

Distributions of Points and Large Quadrangles (Extended Abstract)

Hanno Lefmann

Fakultät für Informatik, TU Chemnitz, D-09107 Chemnitz, Germany
lefmann@informatik.tu-chemnitz.de

Abstract. We consider a variant of Heilbronn's triangle problem by asking, given any integer $n \geq 4$, for the supremum $\Delta_4(n)$ of the minimum area determined by the convex hull of some four of n points in the unit-square $[0, 1]^2$ over all distributions of n points in $[0, 1]^2$. Improving the lower bound $\Delta_4(n) = \Omega(1/n^{3/2})$ of Schmidt [19], we will show that $\Delta_4(n) = \Omega((\log n)^{1/2}/n^{3/2})$ as asked for in [5], by providing a deterministic polynomial time algorithm which finds n points in the unit-square $[0, 1]^2$ that achieve this lower bound.

1 Introduction

The problem of Heilbronn asks for a distribution of n points in the unit-square $[0, 1]^2$ (or unit-ball) such that the minimum area of a triangle determined by three of these n points is as large as possible. Let $\Delta_3(n)$ denote the supremum of this minimum area of a triangle over all distributions of n points in $[0, 1]^2$. In considering for primes n the points $1/n \cdot (i \bmod n, i^2 \bmod n)$, $i = 0, \dots, n-1$, on the moment-curve one easily sees that $\Delta_3(n) = \Omega(1/n^2)$. While for some time this lower bound was believed to be also the upper bound for $\Delta_3(n)$, Komlós, Pintz and Szemerédi [11] showed by probabilistic arguments that $\Delta_3(n) = \Omega(\log n/n^2)$, see Bertram-Kretzberg, Hofmeister and this author [5] for a deterministic polynomial time algorithm achieving this lower bound $\Delta_3(n) = \Omega(\log n/n^2)$. Upper bounds on $\Delta_3(n)$ were given by Roth [14, 15, 16, 17, 18] and Schmidt [19] and, improving these earlier results, the currently best upper bound $\Delta_3(n) = O(2^{c\sqrt{\log n}}/n^{8/7})$, where $c > 0$ is a constant, is due to Komlós, Pintz and Szemerédi [10]. Recently, Jiang, Li and Vitány [9] showed by using methods from Kolmogorov complexity that if n points are distributed uniformly at random and independently of each other in the unit-square $[0, 1]^2$, then the expected value of the minimum area of a triangle formed by some three of these n random points is equal to $\Theta(1/n^3)$. As indicated in [9], this result might be of use to measure the affiancy of certain Monte Carlo methods for determining fair market values of derivatives.

Higher dimensional extensions of Heilbronn's triangle problem were investigated by Barequet [2, 3], who considered for fixed integers $d \geq 2$ the minimum volumes of simplices among n points in the d -dimensional unit-cube $[0, 1]^d$, maximized over all distributions of n points in $[0, 1]^d$, see also [12], [13] and Brass [6].

A variant of Heilbronn’s problem asks, given a fixed integer $k \geq 3$, for the supremum $\Delta_k(n)$ of the minimum area of the convex hull of some k points in a distribution of n points in the unit-square $[0, 1]^2$, where the supremum is over all distributions of n points in $[0, 1]^2$. For $k = 4$, Schmidt [19] proved the nonconstructive lower bound $\Delta_4(n) = \Omega(1/n^{3/2})$. In [5] a deterministic polynomial time algorithm was given, which finds, given an integer $k \geq 3$, for any integer $n \geq k$ a configuration of n points in $[0, 1]^2$ achieving the lower bound $\Delta_k(n) = \Omega(1/n^{(k-1)/(k-2)})$.

Here we provide for $k = 4$ a deterministic polynomial time algorithm which achieves the lower bound $\Delta_4(n) = \Omega((\log n)^{1/2}/n^{3/2})$, as asked for in [5], hence improving also the lower bound $\Delta_4(n) = \Omega(1/n^{3/2})$ of Schmidt [19] by a factor of $\Theta((\log n)^{1/2})$.

Theorem 1. $\dots n \geq 4 \dots \dots \dots [0, 1]^2 \dots \dots n, \dots \dots \Omega((\log n)^{1/2}/n^{3/2})$

Although the mathematics in connection with Heilbronn-type problems seems to be hard, these problems are of interest from the algorithmic point of view as one can test the power of certain algorithmic methods with these problems, i.e. here we will use an approach by approximating the independence number of linear (or uncrowded) hypergraphs. There is some vague indication that perhaps our algorithm finds an asymptotically best possible distribution of points, but at present only $\Delta_4(n) = O(1/n)$ is known.

2 Basic Facts

For distinct points $P, Q \in [0, 1]^2$ let PQ denote the line segment through P and Q . Let $\text{dist}(P, Q)$ denote the distance between the points P and Q . For points $P_1, \dots, P_l \in [0, 1]^2$ let $\text{area}(P_1, \dots, P_l)$ be the area of the convex hull of the points P_1, \dots, P_l . A strip S centered at the line PQ of width w is the set of all points which have distance at most $w/2$ from the line PQ .

Lemma 1. $\dots P_1, \dots, P_l \in [0, 1]^2 \dots \dots \dots (P_1, \dots, P_l) \leq A \dots \dots (P_1, \dots, P_{l-1}) \leq A$

This follows by monotonicity, as by definition $\text{area}(P_1, \dots, P_l)$ is the area of the convex hull of the points P_1, \dots, P_l . □

Lemma 2. $\dots P_1, \dots, P_l \in [0, 1]^2 \dots \dots \dots (P_1, \dots, P_l) \leq A \dots \dots \dots P_i \dots P_j \dots \dots P_k, k \neq i, j, \dots \dots \dots P_i P_j \dots \dots 4 \cdot A / \dots (P_i, P_j)$

Otherwise, by Lemma 1 it is $A \geq \text{area}(P_1, \dots, P_l) \geq \text{area}(P_i, P_j, P_k) > 1/2 \cdot \text{dist}(P_i, P_j) \cdot (2 \cdot A) / \text{dist}(P_i, P_j) = A$, a contradiction. □

Definition 1. $\mathcal{G} = (V, \mathcal{E})$ k $|E| = k$
 $E \in \mathcal{E}$ $\{E, E'\}$ $E, E' \in \mathcal{E}$
 $|E \cap E'| \geq 2$ \mathcal{G} $(2, i)$ $|E \cap E'| = i$
 $i = 2, \dots, k - 1$ \mathcal{G} 2
 $\alpha(\mathcal{G})$ \mathcal{G} $I \subseteq V$
 \mathcal{E} $E \not\subseteq I$ $E \in \mathcal{E}$

3 A Deterministic Algorithm

Here we will prove Theorem 1. By looking at the existing literature, probabilistic existence arguments using evaluations of certain integrals over $[0, 1]^2$ might be possible to get an improvement on $\Delta_4(n) = \Omega(1/n^{3/2})$, however, whether this approach might be successful or not, due to the continuity of the calculations, this does not result in a deterministic algorithm. Therefore, to provide a deterministic polynomial time algorithm, which finds n points in the unit-square $[0, 1]^2$ that achieve the lower bound $\Delta_4(n) = \Omega((\log n)^{1/2}/n^{3/2})$, we will discretize the search space $[0, 1]^2$ by considering the standard $T \times T$ -grid, where $T = n^{1+\alpha}$ for some constant $\alpha > 0$, which will be specified later. However, with this discretization we have to take care of collinear triples of grid-points in the $T \times T$ -grid.

We will transform our problem into a problem of finding in a suitably defined hypergraph a large independent set. For some value $A > 0$, which will be specified later, we form a hypergraph $\mathcal{G} = \mathcal{G}(A) = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$ which contains 3-element and 4-element edges. The vertex set V consists of the T^2 grid-points from the $T \times T$ -grid. The edge sets \mathcal{E}_3 and \mathcal{E}_4 are defined as follows: $\{P, Q, R\} \in \mathcal{E}_3$ if and only if the grid-points $P, Q, R \in V$ lie on a single line, i.e. are collinear, and $\{P, Q, R, S\} \in \mathcal{E}_4$ if and only if no three of the grid-points $P, Q, R, S \in V$ are collinear and area $(P, Q, R, S) \leq A$. In this hypergraph $\mathcal{G} = \mathcal{G}(A)$ we want to find a certain induced subhypergraph $\mathcal{G}^{**} = (V^{**}, \mathcal{E}_4^{**})$ of \mathcal{G} , which does not contain any 3-element edges anymore, hence no three distinct grid-points from the vertex set V^{**} are collinear. An independent set $I \subseteq V^{**}$ in this induced hypergraph $\mathcal{G}^{**} = (V^{**}, \mathcal{E}_4^{**})$ yields $|I|$ grid-points in the $T \times T$ -grid, such that the area of the convex hull of each four distinct grid-points is bigger than A .

An essential tool in our arguments is the following algorithmic version from Bertram-Kretzberg and this author [4] of a deep result of Ajtai, Komlós, Pintz, Spencer and Szemerédi [1], see also [7] and [8].

Theorem 2. $\mathcal{G} = (V, \mathcal{E})$ k
 $t^{k-1} = k \cdot |\mathcal{E}|/|V|$ $\delta > 0$ $O(|V| + |\mathcal{E}| + |V|^3/t^{3-\delta})$
 $I \subseteq V$ $|I| = \Omega(|V|/t \cdot (\log t)^{1/(k-1)})$

The difficulty in our arguments does not lie in the invention of a new algorithm but rather to prove that the algorithm from Theorem 2 can be applied and yields a solution with the desired quality. However, our hypergraph $\mathcal{G} = \mathcal{G}(A)$ is not linear and contains many 2-cycles. The strategy will be to find a certain induced linear subhypergraph \mathcal{G}^{**} of our hypergraph $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$. Therefore, to apply Theorem 2 we will carefully count in the hypergraph $\mathcal{G} = \mathcal{G}(A)$ the numbers $|\mathcal{E}_3|$

and $|\mathcal{E}_4|$ of 3- and 4-element edges, respectively. Also, we will give upper bounds on the numbers of 2-cycles arising from the 4-element edges $E \in \mathcal{E}_4$. Then in a certain induced subhypergraph \mathcal{G}^* of \mathcal{G} we will destroy in one step all induced 3-element edges and all 2-cycles. The resulting induced subhypergraph \mathcal{G}^{**} of \mathcal{G} has not too few vertices and does not contain any 2-cycles anymore and at this point we apply the algorithm from Theorem 2 to \mathcal{G}^{**} .

For positive integers h and s let $\gcd(h, s) \geq 0$ denote the greatest common divisor of h and s . For a grid-point P in the $T \times T$ -grid let p_x and p_y denote its x - and y -coordinate, respectively. We define a partial order \leq_{lex} on the grid-points of the $T \times T$ -grid: for grid-points $P = (p_x, p_y)$ and $Q = (q_x, q_y)$ let

$$P \leq_{lex} Q \iff (p_x < q_x) \text{ or } (p_x = q_x \text{ and } p_y < q_y).$$

Notice that for grid-points $P = (p_x, p_y)$ and $Q = (q_x, q_y)$ with $P \leq_{lex} Q$ there are exactly $(\gcd(q_x - p_x, q_y - p_y) - 1)$ grid-points on the segment $[P, Q]$ excluding the endpoints P and Q .

We will use the following result from [5].

Lemma 3. Let $P = (p_x, p_y)$ and $R = (r_x, r_y)$ be grid-points in the $T \times T$ -grid with $P \leq_{lex} R$ and $s := r_x - p_x \geq 0$.

- (i) The number of grid-points Q in the $T \times T$ -grid with $P \leq_{lex} Q \leq_{lex} R$ is at most $4 \cdot A$.
- (ii) The number of collinear triples (P, Q, R) is at most $12 \cdot A \cdot T/s$ if $s > 0$ and $4 \cdot A \cdot T$ if $s = 0$.

First we will estimate the numbers $|\mathcal{E}_3|$ and $|\mathcal{E}_4|$ of 3- and 4-element edges, respectively, in the hypergraph $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$.

Lemma 4. Let $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$ be a hypergraph with $|\mathcal{E}_3| \leq 3$.

$$|\mathcal{E}_3| \leq c_3 \cdot T^4 \cdot \log T. \tag{1}$$

We remark that in [5] an upper bound of $O(T^{4+\epsilon})$, for any $\epsilon > 0$, on the number of collinear triples of grid-points in the $T \times T$ -grid was proved.

Let P, Q, R be grid-points from the $T \times T$ -grid, where $P \leq_{lex} Q \leq_{lex} R$. Set $s := r_x - p_x \geq 0$ and $h := r_y - p_y$. We have $\{P, Q, R\} \in \mathcal{E}_3$ if and only if P, Q, R are collinear. For $h = 0$ or $s = 0$ the number of collinear triples P, Q, R of grid-points is at most $O(T^4)$, as we can choose one of the $2 \cdot T$ horizontal or vertical lines and on each of these at most $O(T^3)$ triples of grid-points.

Let $h, s \neq 0$. A grid-point P can be chosen in at most T^2 ways. Given the grid-point P , any grid-point R is determined by a pair (s, h) of integers with $1 \leq h \leq s \leq T$. Those pairs (s, h) of integers with $0 \leq -h \leq s \leq T$ or

$1 \leq s < |h| \leq T$ will be taken into account by an additional constant factor using rotation symmetry.

On the segment $[P, R]$ there are at most $\gcd(h, s)$ grid-points Q excluding P and R . Thus, for a constant $c' > 0$ the number of collinear triples of grid-points in the $T \times T$ -grid is at most

$$c' \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=1}^s \gcd(h, s).$$

Each divisor d of s divides at most s/d positive integers x with $x \leq s$, namely the integers $i \cdot d, i = 1, \dots, [s/d]$, hence we infer for a constant $c_3 > 0$:

$$\begin{aligned} c' \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=1}^s \gcd(h, s) &\leq c' \cdot T^2 \cdot \sum_{s=1}^T \sum_{d|s} \frac{s}{d} \cdot d \leq c' \cdot T^2 \cdot \sum_{s=1}^T s \sum_{d|s} 1 \leq \\ &\leq c' \cdot T^2 \cdot \sum_{d=1}^T \sum_{i=1}^{[T/d]} i \cdot d \leq c' \cdot T^2 \cdot \sum_{d=1}^T d \cdot \frac{T^2}{d^2} = c' \cdot T^4 \cdot \sum_{d=1}^T \frac{1}{d} \leq \\ &\leq c_3 \cdot T^4 \cdot \log T. \end{aligned} \quad \square$$

Lemma 5. $|\mathcal{E}_4| \leq c_4 \cdot A^2 \cdot T^4$ for the number of 4-element edges $\{P_1, P_2, P_3, P_4\}$ in the $T \times T$ -grid with $\text{area}(P_1, P_2, P_3, P_4) \leq A$.

$$|\mathcal{E}_4| \leq c_4 \cdot A^2 \cdot T^4. \tag{2}$$

We can assume that $P_1 \leq_{lex} P_3 \leq_{lex} P_4 \leq_{lex} P_2$. Let $s := p_{2,x} - p_{1,x} \geq 0$ and $h := p_{2,y} - p_{1,y}$. If $s = 0$, then the grid-points P_1, P_2, P_3, P_4 are collinear, hence we have $s \neq 0$. By rotation symmetry, which we take into account by an additional constant factor, we can assume that $0 \leq h \leq s \leq T$.

If $\text{area}(P_1, P_2, P_3, P_4) \leq A$, then by Lemma 2 we have $\text{area}(P_1, P_2, P_3) \leq A$ and $\text{area}(P_1, P_2, P_4) \leq A$. There are T^2 choices for the grid-point P_1 . Given the grid-point P_1 , any grid-point P_2 is determined by a pair $(s, h) \neq (0, 0)$ of integers with $0 \leq h \leq s \leq T$. Since $P_1 \leq_{lex} P_3 \leq_{lex} P_4 \leq_{lex} P_2$ and since no three of the grid-points P_1, P_2, P_3, P_4 are collinear, by Lemma 3(a) there are at most $4 \cdot A$ choices for the grid-points P_3 and P_4 each. Thus, we obtain for constants $c', c_4 > 0$ for the number $|\mathcal{E}_4|$ of 4-element edges in the hypergraph \mathcal{G} an upper bound of

$$|\mathcal{E}_4| \leq c' \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s (4 \cdot A)^2 \leq 16 \cdot c' \cdot A^2 \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s 1 \leq c_4 \cdot A^2 \cdot T^4. \quad \square$$

By (2) the average degree t^3 of the hypergraph $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$ for the 4-element edges $E \in \mathcal{E}_4$ satisfies

$$t^3 = \frac{4 \cdot |\mathcal{E}_4|}{|V|} \leq \frac{4 \cdot c_4 \cdot A^2 \cdot T^4}{T^2} = 4 \cdot c_4 \cdot A^2 \cdot T^2 := t_0^3. \tag{3}$$

3.1 (2, 2)- and (2, 3)-Cycles

Let $s_{2,i}(\mathcal{G}; \mathcal{E}_4)$ denote the number of $(2, i)$ -cycles, $i = 2, 3$, formed by unordered pairs of 4-element edges in the hypergraph $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$. Here we will give upper bounds on the numbers of these $(2, 2)$ - and $(2, 3)$ -cycles in \mathcal{G} .

Lemma 6. $s_{2,3}(\mathcal{G}; \mathcal{E}_4) = O(A^3 \cdot T^4 \cdot \log T)$ for $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$.

$$s_{2,3}(\mathcal{G}; \mathcal{E}_4) = O(A^3 \cdot T^4 \cdot \log T). \tag{4}$$

Assume that the quadruples P_1, P_2, P_3, P_4 and P_1, P_2, P_3, P_5 of grid-points determine a $(2, 3)$ -cycle in the hypergraph \mathcal{G} , where no three of these five grid-points are collinear. Then $\text{area}(P_1, P_2, P_3, P_4) \leq A$ and $\text{area}(P_1, P_2, P_3, P_5) \leq A$. We can assume that $P_1 \leq_{lex} P_3 \leq_{lex} P_2$. There are T^2 possibilities for the grid-point P_1 . Let $s := p_{2,x} - p_{1,x}$ and $h := p_{2,y} - p_{1,y}$. Given the grid-point P_1 , any grid-point P_2 is determined by a pair $(s, h) \neq (0, 0)$ of integers, where by rotation symmetry w.l.o.g. $0 \leq h \leq s \leq T$. Clearly, we have $s > 0$, as for $s = 0$ the grid-points P_1, P_2, P_3 are collinear. Since $\text{area}(P_1, P_2, P_3) \leq A$ by Lemma 1, and since P_1, P_2, P_3 are not collinear, by Lemma 3(a) there are at most $4 \cdot A$ choices for the grid-point P_3 .

With $\text{area}(P_1, P_2, P_4) \leq A$ and $\text{area}(P_1, P_2, P_5) \leq A$, and since P_1, P_2, P_4 and P_1, P_2, P_5 are not collinear, by Lemma 3(b) there are at most $12 \cdot A \cdot T/s$ choices for each grid-point P_4 and P_5 , hence for some constants $c', c'', c''', c_{2,3} > 0$ we have the following upper bound on the number $s_{2,3}(\mathcal{G}; \mathcal{E}_4)$ of $(2, 3)$ -cycles in \mathcal{G} :

$$s_{2,3}(\mathcal{G}; \mathcal{E}_4) \leq c' \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s (4 \cdot A) \cdot \left(\frac{12 \cdot A \cdot T}{s}\right)^2 \leq c'' \cdot A^3 \cdot T^4 \cdot \sum_{s=1}^T \sum_{h=0}^s \frac{1}{s^2} \leq c''' \cdot A^3 \cdot T^4 \cdot \sum_{s=1}^T \frac{1}{s} \leq c_{2,3} \cdot A^3 \cdot T^4 \cdot \log T. \quad \square$$

Next we will estimate the number of $(2, 2)$ -cycles in the hypergraph \mathcal{G} .

Lemma 7. $s_{2,2}(\mathcal{G}; \mathcal{E}_4) = O(A^4 \cdot T^{9/2})$ for $\mathcal{G} = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$.

$$s_{2,2}(\mathcal{G}; \mathcal{E}_4) = O(A^4 \cdot T^{9/2}). \tag{5}$$

Let us denote the grid-points of two 4-element edges $E, E' \in \mathcal{E}_4$, which yield a $(2, 2)$ -cycle in the hypergraph \mathcal{G} , by P_1, P_2, P_3, P_4 and P_1, P_2, P_5, P_6 , where $P_1 \leq_{lex} P_2$ and no three of the grid-points of these edges are collinear. Let $u := \lceil T^\gamma \rceil$ where $0 < \gamma < 1$ is a constant, which will be specified later.

There are T^2 choices for the grid-point P_1 . Given the grid-point P_1 , any grid-point P_2 is determined by a pair $(s, h) \neq (0, 0)$ of integers. By rotation symmetry we can assume that $s > 0$ and $0 \leq h \leq s \leq T$. By Lemma 2 all grid-points P_3, P_4, P_5, P_6 must lie in a strip S centered at the line P_1P_2 of width $4 \cdot A/\sqrt{h^2 + s^2}$.

Consider a parallelogram $\mathcal{P}_0 = \{(p_x, p_y) \in S \mid p_{1,x} - u \leq_{lex} p_x \leq_{lex} p_{2,x} + u\}$ with two of its boundaries being the boundaries of the strip S and with center being the middle of the segment $[P_1, P_2]$. By Lemma 3(a) this parallelogram \mathcal{P}_0 contains at most $4 \cdot A \cdot (s+2 \cdot u)/s$ grid-points P , where P_1, P_2, P are not collinear. In the following we will distinguish whether some of the grid-points P_3, P_4, P_5, P_6 lie in the parallelogram \mathcal{P}_0 or not.

All grid-points P_3, P_4, P_5, P_6 satisfy $P_3, P_4, P_5, P_6 \in \mathcal{P}_0$.

Given the grid-points P_1 and P_2 , there are at most $4 \cdot A \cdot (s+2 \cdot u)/s$ choices for each of the grid-points $P_3, P_4, P_5, P_6 \in \mathcal{P}_0$. Using $u = \lceil T^\gamma \rceil$, for some constants $c, c', c'', c''' > 0$ we obtain for the number of these (2, 2)-cycles arising from \mathcal{E}_4 the following upper bound

$$\begin{aligned} & c \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s \left(\frac{4 \cdot A \cdot (s+2 \cdot u)}{s} \right)^4 \\ & \leq c' \cdot A^4 \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s \left(1 + \frac{2 \cdot u}{s} \right)^4 \\ & \leq c' \cdot A^4 \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s \left(1 + \frac{8 \cdot u}{s} + \frac{24 \cdot u^2}{s^2} + \frac{32 \cdot u^3}{s^3} + \frac{16 \cdot u^4}{s^4} \right) \\ & \leq c'' \cdot A^4 \cdot T^2 \cdot (T^2 + T^{1+\gamma} + T^{2\gamma} \cdot \log T + T^{3\gamma} + T^{4\gamma}) \\ & \leq c''' \cdot A^4 \cdot (T^4 + T^{2+4\gamma}) \qquad \text{as } 0 < \gamma < 1 \text{ is a constant.} \end{aligned} \tag{6}$$

The grid-points P_3, P_4, P_5 satisfy $P_3, P_4 \in \mathcal{P}_0$ and $P_5 \notin \mathcal{P}_0$.

Given the grid-points P_1 and P_2 , there are at most $4 \cdot A \cdot (s+2 \cdot u)/s$ choices for each of the grid-points $P_3 \in \mathcal{P}_0$ and $P_4 \in \mathcal{P}_0$. Considering now the quadruple P_1, P_2, P_5, P_6 of grid-points, by Lemma 3(b) there are at most $12 \cdot A \cdot T/s$ choices for the grid-point $P_5 = (p_{5,x}, p_{5,y}) \notin \mathcal{P}_0$, since P_1, P_2, P_5 are not collinear and area $(P_1, P_2, P_5) \leq A$. However, since now $|p_{1,x} - p_{5,x}| \geq u$ and area $(P_1, P_2, P_6) \leq A$, by Lemma 3(b) the number of choices for the grid-point P_6 in the $T \times T$ -grid, such that P_1, P_2, P_6 are not collinear, is at most $12 \cdot A \cdot T/u$. With $u = \lceil T^\gamma \rceil$ we obtain for constants $c', c'', c''' > 0$ the following upper bound on the number of these (2, 2)-cycles arising from \mathcal{E}_4 :

$$\begin{aligned} & c \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s \frac{12 \cdot A \cdot T}{u} \cdot \frac{12 \cdot A \cdot T}{s} \cdot \left(\frac{4 \cdot A \cdot (s+2 \cdot u)}{s} \right)^2 \\ & \leq c' \cdot A^4 \cdot T^4 \cdot \sum_{s=1}^T \sum_{h=0}^s \left(\frac{1}{s \cdot u} + \frac{4}{s^2} + \frac{4 \cdot u}{s^3} \right) \\ & \leq c'' \cdot A^4 \cdot T^4 \cdot \sum_{s=1}^T \left(\frac{1}{T^\gamma} + \frac{4}{s} + \frac{4 \cdot T^\gamma}{s^2} \right) \\ & \leq c''' \cdot A^4 \cdot (T^{5-\gamma} + T^{4+\gamma}) \qquad \text{as } 0 < \gamma < 1 \text{ is a constant.} \end{aligned} \tag{7}$$

The grid-points P_3, P_5 satisfy $P_3, P_5 \notin \mathcal{P}_0$.

Given the grid-points P_1 and P_2 , we partition the strip S within the $T \times T$ -grid into parallelograms $\mathcal{P}_0, \mathcal{P}_i^+, \mathcal{P}_i^-, i = 1, \dots, l \leq T^{1-\gamma}$, which are all translates of \mathcal{P}_0 , and arranged according to the order $\mathcal{P}_l^+, \mathcal{P}_{l-1}^+, \dots, \mathcal{P}_1^+, \mathcal{P}_0, \mathcal{P}_1^-, \dots, \mathcal{P}_{l-1}^-, \mathcal{P}_l^-$. Each grid-point $P = (p_x, p_y) \in \mathcal{P}_i^+ \cup \mathcal{P}_i^-, i \geq 1$, satisfies $|p_x - p_{1,x}| \geq s + u + (i - 1) \cdot (s + 2 \cdot u) \geq i \cdot (s + u)$ or $|p_x - p_{2,x}| \geq i \cdot (s + u)$. Using Lemma 3(a), each parallelogram \mathcal{P}_i^+ or \mathcal{P}_i^- contains at most $4 \cdot A \cdot (s + 2 \cdot u)/s$ grid-points P , such that P_1, P_2, P are not collinear. Each grid-point P_3, P_5 lies in some parallelogram \mathcal{P}_i^+ or \mathcal{P}_i^- for some $i \geq 1$. If $P_3 \in \mathcal{P}_i^+ \cup \mathcal{P}_i^-, i \geq 1$, then by Lemma 3(b) there are at most $12 \cdot A \cdot T/(i \cdot (s + u))$ choices for the grid-point P_4 . Similarly, if $P_5 \in \mathcal{P}_j^+ \cup \mathcal{P}_j^-, j \geq 1$, there are at most $12 \cdot A \cdot T/(j \cdot (s + u))$ choices for the grid-point P_6 , and we obtain for constants $c, c', c'', c''' > 0$ for the number of these $(2, 2)$ -cycles among \mathcal{E}_4 the upper bound

$$\begin{aligned} & c \cdot T^2 \cdot \sum_{s=1}^T \sum_{h=0}^s \sum_{i=1}^l \sum_{j=1}^l \left(\frac{4 \cdot A \cdot (s + 2 \cdot u)}{s} \right)^2 \cdot \frac{(12 \cdot A \cdot T)^2}{i \cdot j \cdot (s + u)^2} \\ & \leq c' \cdot A^4 \cdot T^4 \cdot \sum_{s=1}^T \sum_{h=0}^s \left(\frac{s + 2 \cdot u}{s} \right)^2 \cdot \left(\frac{1}{s + u} \right)^2 \cdot \sum_{i=1}^l \frac{1}{i} \sum_{j=1}^l \frac{1}{j} \\ & \leq c'' \cdot A^4 \cdot T^4 \cdot (\log T)^2 \cdot \sum_{s=1}^T \sum_{h=0}^s \frac{4}{s^2} \\ & \leq c''' \cdot A^4 \cdot T^4 \cdot (\log T)^3, \end{aligned} \tag{8}$$

where we used $(s + 2 \cdot u)/(s \cdot (s + u)) \leq 2/s$ for $s, u \geq 0$.

For $\gamma := 1/2$ the estimates (6),(7), (8) yield $s_{2,2}(\mathcal{G}; \mathcal{E}_4) = O(A^4 \cdot T^{9/2})$. □

3.2 Selecting a Subhypergraph

For a suitable constant $c > 0$ we set

$$A := \frac{c \cdot T^2 \cdot (\log n)^{1/2}}{n^{3/2}} > 1. \tag{9}$$

For a moment we will use a probabilistic argument to simplify the presentation. However, this argument will be made constructive shortly. With probability $p := T^\epsilon/t_0 \leq 1$ for a small constant $\epsilon > 0$ we pick uniformly at random and independently of each other vertices from the set V . Let $V^* \subseteq V$ be the resulting random subset of V of the picked vertices and let $\mathcal{G}^* = (V^*, \mathcal{E}_3^* \cup \mathcal{E}_4^*)$ with $\mathcal{E}_3^* := \mathcal{E}_3 \cap [V^*]^3$ and $\mathcal{E}_4^* := \mathcal{E}_4 \cap [V^*]^4$ be the resulting random induced subhypergraph of \mathcal{G} . Let $E[|V^*|], E[|\mathcal{E}_3^*|], E[|\mathcal{E}_4^*|], E[s_{2,2}(\mathcal{G}^*; \mathcal{E}_4^*)], E[s_{2,3}(\mathcal{G}^*; \mathcal{E}_4^*)]$ denote the expected number of vertices, 3-element edges, 4-element edges, $(2, 2)$ - and $(2, 3)$ -cycles arising from \mathcal{E}_4^* , respectively, in the random subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_3^* \cup \mathcal{E}_4^*)$ of \mathcal{G} . By (1), (2), (5) and (4) we infer for constants $c'_1, c'_3, c'_4, c'_{2,2}, c'_{2,3} > 0$:

$$E[|V^*|] = p \cdot T^2 = c'_1 \cdot T^{4/3+\varepsilon} / A^{2/3} \tag{10}$$

$$\begin{aligned} E[|\mathcal{E}_3^*|] &= p^3 \cdot |\mathcal{E}_3| \leq c'_3 \cdot (T^4 \cdot \log T) \cdot T^{3\varepsilon} / (A \cdot T)^2 \leq \\ &\leq c'_3 \cdot T^{2+3\varepsilon} \cdot \log T / A^2 \end{aligned} \tag{11}$$

$$\begin{aligned} E[|\mathcal{E}_4^*|] &= p^4 \cdot |\mathcal{E}_4| \leq c'_4 \cdot (A^2 \cdot T^4) \cdot T^{4\varepsilon} / (A \cdot T)^{8/3} \leq \\ &\leq c'_4 \cdot T^{4/3+4\varepsilon} / A^{2/3} \end{aligned} \tag{12}$$

$$\begin{aligned} E[s_{2,2}(\mathcal{G}^*; \mathcal{E}_4^*)] &\leq p^6 \cdot s_{2,2}(\mathcal{G}; \mathcal{E}_4) \leq c'_{2,2} \cdot (A^4 \cdot T^{9/2}) \cdot T^{6\varepsilon} / (A \cdot T)^4 \leq \\ &\leq c'_{2,2} \cdot T^{1/2+6\varepsilon} \end{aligned} \tag{13}$$

$$\begin{aligned} E[s_{2,3}(\mathcal{G}^*; \mathcal{E}_4^*)] &\leq p^5 \cdot s_{2,3}(\mathcal{G}; \mathcal{E}_4) \leq c'_{2,3} \cdot (A^3 \cdot T^4 \cdot \log T) \cdot T^{5\varepsilon} / (A \cdot T)^{10/3} \leq \\ &\leq c'_{2,3} \cdot T^{2/3+5\varepsilon} \cdot \log T / A^{1/3} . \end{aligned} \tag{14}$$

With (10), (11), (12), (13), (14) and Chernoff's and Markov's inequality there exists a subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_3^* \cup \mathcal{E}_4^*)$ of \mathcal{G} such that

$$|V^*| \geq c'_1 / 2 \cdot T^{4/3+\varepsilon} / A^{2/3} \tag{15}$$

$$|\mathcal{E}_3^*| \leq 5 \cdot c'_3 \cdot T^{2+3\varepsilon} \cdot \log T / A^2 \tag{16}$$

$$|\mathcal{E}_4^*| \leq 5 \cdot c'_4 \cdot T^{4/3+4\varepsilon} / A^{2/3} \tag{17}$$

$$s_{2,2}(\mathcal{G}^*; \mathcal{E}_4^*) \leq 5 \cdot c'_{2,2} \cdot T^{1/2+6\varepsilon} \tag{18}$$

$$s_{2,3}(\mathcal{G}^*; \mathcal{E}_4^*) \leq 5 \cdot c'_{2,3} \cdot T^{2/3+5\varepsilon} \cdot \log T / A^{1/3} . \tag{19}$$

This probabilistic argument can be derandomized by using the method of conditional probabilities as follows. For $j = 2, 3$, let \mathcal{C}_j be the (multi-)set of all $(8 - j)$ -element subsets $E \cup E'$ of V such that the pair $\{E, E'\}$ of distinct 4-element edges $E, E' \in \mathcal{E}_4$ yields a $(2, j)$ -cycle in \mathcal{G} . We enumerate the vertices of the $T \times T$ -grid by P_1, \dots, P_{T^2} . For each vertex P_i we associate a parameter $p_i \in [0, 1]$, and we define a potential function F by

$$\begin{aligned} F(p_1, \dots, p_{T^2}) &:= 2^{p \cdot T^2 / 2} \cdot \prod_{i=1}^{T^2} \left(1 - \frac{p_i}{2} \right) + \frac{\sum_{\{i,j,k\} \in \mathcal{E}_3} p_i \cdot p_j \cdot p_k}{5 \cdot c'_3 \cdot T^{2+3\varepsilon} \cdot \log T / A^2} + \\ &+ \frac{\sum_{\{i,j,k,l\} \in \mathcal{E}_4} p_i \cdot p_j \cdot p_k \cdot p_l}{5 \cdot c'_4 \cdot T^{4/3+4\varepsilon} / A^{2/3}} + \frac{\sum_{\{i,j,k,l,m,o\} \in \mathcal{C}_2} p_i \cdot p_j \cdot p_k \cdot p_l \cdot p_m \cdot p_o}{5 \cdot c'_{2,2} \cdot T^{1/2+6\varepsilon}} + \\ &+ \frac{\sum_{\{i,j,k,l,m\} \in \mathcal{C}_3} p_i \cdot p_j \cdot p_k \cdot p_l \cdot p_m}{5 \cdot c'_{2,3} \cdot T^{2/3+5\varepsilon} \cdot \log T / A^{1/3}} . \end{aligned}$$

With the initialisation $p_1 := \dots := p_{T^2} := p := T^\varepsilon / t_0$, we infer $F(p, \dots, p) < (2/e)^{pT^2/2} + 4/5 < 1$ for $p \cdot T^2 \geq 11$. Then, using the linearity of $F(p_1, \dots, p_{T^2})$ in each p_i , we minimize $F(p_1, \dots, p_{T^2})$ by choosing one after the other $p_i := 0$ or $p_i := 1$ for $i = 1, \dots, T^2$, and finally we obtain $F(p_1, \dots, p_{T^2}) < 1$. Then the vertex set $V^* = \{i \in V \mid p_i = 1\}$ yields an induced subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_3^* \cup \mathcal{E}_4^*)$ of \mathcal{G} with $\mathcal{E}_3^* = \mathcal{E}_3 \cup [V^*]^3$ and $\mathcal{E}_4^* = \mathcal{E}_4 \cup [V^*]^4$, which satisfies (15), (16), (17), (18), (19), compare [4]. With (1), (2), (4), (5), (9) and using that $A > 1$, the running time of this derandomization is given by

$$O(|V| + |\mathcal{E}_3| + |\mathcal{E}_4| + |\mathcal{C}_2| + |\mathcal{C}_3|) = O(A^4 \cdot T^{\frac{9}{2}}) = O(T^{\frac{25}{2}} \cdot (\log n)^2/n^6). \tag{20}$$

We will show next that the numbers $|\mathcal{E}_3^*|$, $s_{2,2}(\mathcal{G}^*; \mathcal{E}_4^*)$ and $s_{2,3}(\mathcal{G}^*; \mathcal{E}_4^*)$ of 3-element edges, (2, 2)- and (2, 3)-cycles arising from \mathcal{E}_4^* in \mathcal{G}^* , respectively, are very small compared to the number $|V^*|$ of vertices in \mathcal{G}^* .

Lemma 8. $0 < \varepsilon < \alpha/(1 + \alpha)$. . .

$$|\mathcal{E}_3^*| = o(|V^*|) . \tag{21}$$

By (9), (15), (16) and using $T = n^{1+\alpha}$ with $\alpha > 0$ a constant we have

$$\begin{aligned} |\mathcal{E}_3^*| = o(|V^*|) &\iff T^{2+3\varepsilon} \cdot \log T/A^2 = o(T^{4/3+\varepsilon}/A^{2/3}) \iff \\ &\iff T^{2/3+2\varepsilon} \cdot \log T/A^{4/3} = o(1) \iff \frac{n^2 \cdot \log T}{T^{2-2\varepsilon} \cdot (\log n)^{2/3}} = o(1) \iff \\ &\iff n^{2-(1+\alpha)(2-2\varepsilon)} \cdot (\log n)^{1/3} = o(1) \iff (1 + \alpha) \cdot (2 - 2\varepsilon) > 2 \iff \\ &\iff \varepsilon < \alpha/(1 + \alpha) . \quad \square \end{aligned}$$

Lemma 9. $0 < \varepsilon < 1/(5 \cdot (1 + \alpha)) - 1/10$. . .

$$s_{2,2}(\mathcal{G}^*; \mathcal{E}_4^*) = o(|V^*|) . \tag{22}$$

By (9), (15), (18) and using $T = n^{1+\alpha}$ with $\alpha > 0$ a constant we infer

$$\begin{aligned} s_{2,2}(\mathcal{G}^*; \mathcal{E}_4^*) = o(|V^*|) &\iff T^{1/2+6\varepsilon} = o(T^{4/3+\varepsilon}/A^{2/3}) \iff \\ &\iff A^{2/3}/T^{5/6-5\varepsilon} = o(1) \iff n^{(1+\alpha)(1/2+5\varepsilon)-1} \cdot (\log n)^{1/3} = o(1) \iff \\ &\iff (1 + \alpha) \cdot (1/2 + 5 \cdot \varepsilon) < 1 \iff \varepsilon < 1/(5 \cdot (1 + \alpha)) - 1/10 . \quad \square \end{aligned}$$

Lemma 10. $0 < \varepsilon < 1/(8 \cdot (1 + \alpha))$. . .

$$s_{2,3}(\mathcal{G}^*; \mathcal{E}_4^*) = o(|V^*|) . \tag{23}$$

By (9), (15), (19) and using $T = n^{1+\alpha}$ with $\alpha > 0$ a constant we have

$$\begin{aligned} s_{2,3}(\mathcal{G}^*; \mathcal{E}_4^*) = o(|V^*|) &\iff T^{2/3+5\varepsilon} \cdot \log T/A^{1/3} = o(T^{4/3+\varepsilon}/A^{2/3}) \iff \\ &\iff A^{1/3} \cdot \log T/T^{2/3-4\varepsilon} = o(1) \iff n^{(1+\alpha)4\varepsilon-1/2} \cdot (\log n)^{7/6} = o(1) \iff \\ &\iff (1 + \alpha) \cdot 4 \cdot \varepsilon < 1/2 \iff \varepsilon < 1/(8 \cdot (1 + \alpha)) . \quad \square \end{aligned}$$

To satisfy $p = T^\varepsilon/t_0 \leq 1$, we need $T^\varepsilon/((4 \cdot c_4)^{1/3} \cdot A^{2/3} \cdot T^{2/3}) \leq 1$, which holds by (9) for $0 < \varepsilon \leq 2 - 1/(1 + \alpha)$. For $\varepsilon := 1/(12 \cdot (1 + \alpha))$ and $1/12 < \alpha < 1/6$ all assumptions in Lemmas 8, 9, 10 are fulfilled. We delete in $\mathcal{G}^* = (V^*, \mathcal{E}_3^* \cup \mathcal{E}_4^*)$ one vertex from each 3-element edge $E \in \mathcal{E}_3^*$, and from each (2, 2)- and (2, 3)-cycle in \mathcal{G}^* arising from \mathcal{E}_4^* , and we obtain a subset $V^{**} \subseteq V^*$ with $|V^{**}| = (1 - o(1)) \cdot |V^*|$. The on V^{**} induced subhypergraph \mathcal{G}^{**} of \mathcal{G}^* contains no 3-element edges or

(2, 2)- or (2, 3)-cycles, i.e. $\mathcal{G}^{**} = (V^{**}, \mathcal{E}_4^{**})$ with $\mathcal{E}_4^{**} = \mathcal{E}_4^* \cap [V^{**}]^4$ is a linear and 4-uniform hypergraph. By (15) and (17) we have

$$\begin{aligned} |V^{**}| &\geq (c'_1/2 - o(1)) \cdot T^{4/3+\varepsilon} / A^{2/3} \\ |\mathcal{E}_4^{**}| &\leq 5 \cdot c'_4 \cdot T^{4/3+4\varepsilon} / A^{2/3}, \end{aligned}$$

hence the average degree t^3 of the subhypergraph $\mathcal{G}^{**} = (V^{**}, \mathcal{E}_4^{**})$ satisfies

$$t^3 = \frac{|\mathcal{E}_4^{**}|}{|V^{**}|} \leq \frac{20 \cdot c'_4 \cdot T^{4/3+4\varepsilon} / A^{2/3}}{(c'_1/2 - o(1)) \cdot T^{4/3+\varepsilon} / A^{2/3}} = \frac{40 \cdot c'_4}{c'_1 - o(1)} \cdot T^{3\varepsilon} := t_1^3.$$

Since \mathcal{G}^{**} is linear we can now apply Theorem 2 and, using (9), we find in time

$$O\left(|V^{**}| + |\mathcal{E}_4^{**}| + \frac{|V^{**}|^3}{t_1^{3-\delta}}\right) = O\left(\frac{T^{4/3+4\varepsilon}}{A^{2/3}} + \frac{T^{4+\varepsilon\delta}}{A^2}\right) = O\left(\frac{n^{3+\delta/12}}{\log n}\right) \tag{24}$$

for any $\delta > 0$, an independent set I of size

$$\begin{aligned} |I| &= \Omega\left(\frac{|V^{**}|}{t} \cdot (\log t)^{\frac{1}{3}}\right) = \Omega\left(\frac{|V^{**}|}{t_1} \cdot (\log t_1)^{\frac{1}{3}}\right) = \\ &= \Omega\left(\frac{T^{4/3+\varepsilon} / A^{2/3}}{T^\varepsilon} \cdot (\log T^\varepsilon)^{\frac{1}{3}}\right) = \Omega\left(\frac{T^{4/3}}{A^{2/3}} \cdot (\log T)^{\frac{1}{3}}\right) = \\ &= \Omega\left(\frac{n}{(\log n)^{\frac{1}{3}}} \cdot (\log T)^{\frac{1}{3}}\right) = \Omega(n) \quad \text{since } T = n^{1+\alpha} \text{ and } \alpha > 0 \text{ is constant.} \end{aligned}$$

By choosing a sufficiently small constant $c > 0$ in (9), we obtain in \mathcal{G}^{**} and hence in \mathcal{G} an independent set of size n , which yields a desired set of n grid-points in the $T \times T$ -grid such that the area of the convex hull of every four distinct of these n points is at least $A = \Theta(T^2 \cdot (\log n)^{1/2} / n^{3/2})$. After rescaling we have n points in $[0, 1]^2$ such that the area of the convex hull of every four distinct of these n points is at least $\Omega((\log n)^{1/2} / n^{3/2})$. In comparing the running times (20) and (24) we get the overall time bound $O(T^{25/2} \cdot (\log n)^2 / n^6)$ for $0 < \delta < 1$. For $\alpha = 1/11$, say, we have the time bound $O(n^{84/11} \cdot (\log n)^2) = o(n^8)$. Indeed, we achieve the time bound $O(n^{13/2+\delta})$ for any $\delta > 0$ by choosing $\varepsilon := 1/(C \cdot (1 + \alpha))$ and $\alpha := 1/(C - 1)$, where $C \geq 12$ is a large enough constant, i.e. $C > 1 + 25/(2 \cdot \delta)$.

4 Final Remarks

It seems that our approach also works for improving algorithmically the lower bound on $\Delta_k(n)$ from [5] for $k = 5$ by a poly-logarithmic factor, but not for arbitrary values $k \geq 6$. However, this is work in progress. To decide, whether our algorithm yields optimal solutions, one needs an upper bound. However, at present only $\Delta_k(n) = O(1/n)$ for fixed integers $k \geq 4$ is known, see [19].

References

1. M. Ajtai, J. Komlós, J. Pintz, J. Spencer and E. Szemerédi, *Extremal Uncrowded Hypergraphs*, Journal of Combinatorial Theory Ser. A, 32, 1982, 321–335.
2. G. Barequet, *A Lower Bound for Heilbronn's Triangle Problem in d Dimensions*, SIAM Journal on Discrete Mathematics 14, 2001, 230–236.
3. G. Barequet, *The On-Line Heilbronn's Triangle Problem in Three and Four Dimensions*, Proceedings COCOON'02', LNCS 2387, Springer, 2002, 360–369.
4. C. Bertram–Kretzberg and H. Lefmann, *The Algorithmic Aspects of Uncrowded Hypergraphs*, SIAM Journal on Computing 29, 1999, 201–230.
5. C. Bertram–Kretzberg, T. Hofmeister and H. Lefmann, *An Algorithm for Heilbronn's Problem*, SIAM Journal on Computing 30, 2000, 383–390.
6. P. Brass, *An Upper Bound for the d -Dimensional Heilbronn Triangle Problem*, preprint, 2003.
7. R. A. Duke, H. Lefmann and V. Rödl, *On Uncrowded Hypergraphs*, Random Structures & Algorithms 6, 1995, 209–212.
8. A. Fundia, *Derandomizing Chebychev's Inequality to find Independent Sets in Uncrowded Hypergraphs*, Random Structures & Algorithms, 8, 1996, 131–147.
9. T. Jiang, M. Li and P. Vitany, *The Average Case Area of Heilbronn-type Triangles*, Random Structures & Algorithms 20, 2002, 206–219.
10. J. Komlós, J. Pintz and E. Szemerédi, *On Heilbronn's Triangle Problem*, Journal of the London Mathematical Society, 24, 1981, 385–396.
11. J. Komlós, J. Pintz and E. Szemerédi, *A Lower Bound for Heilbronn's Problem*, Journal of the London Mathematical Society, 25, 1982, 13–24.
12. H. Lefmann, *On Heilbronn's Problem in Higher Dimension*, Combinatorica 23, 2003, 669–680.
13. H. Lefmann and N. Schmitt, *A Deterministic Polynomial Time Algorithm for Heilbronn's Problem in Three Dimensions*, SIAM Journal on Computing 31, 2002, 1926–1947.
14. K. F. Roth, *On a Problem of Heilbronn*, Journal of the London Mathematical Society 26, 1951, 198–204.
15. K. F. Roth, *On a Problem of Heilbronn, II*, Proc. of the London Mathematical Society (3), 25, 1972, 193–212.
16. K. F. Roth, *On a Problem of Heilbronn, III*, Proc. of the London Mathematical Society (3), 25, 1972, 543–549.
17. K. F. Roth, *Estimation of the Area of the Smallest Triangle Obtained by Selecting Three out of n Points in a Disc of Unit Area*, Proc. of Symposia in Pure Mathematics, 24, 1973, AMS, Providence, 251–262.
18. K. F. Roth, *Developments in Heilbronn's Triangle Problem*, Advances in Mathematics, 22, 1976, 364–385.
19. W. M. Schmidt, *On a Problem of Heilbronn*, Journal of the London Mathematical Society (2), 4, 1972, 545–550.

Cutting Out Polygons with Lines and Rays^{*}

Ovidiu Daescu and Jun Luo

Department of Computer Science, University of Texas at Dallas,
Richardson, TX 75080, USA
{daescu, ljroger}@utdallas.edu

Abstract. We present approximation algorithms for cutting out polygons with line cuts and ray cuts. Our results answer a number of open problems and are either the first solutions or significantly improve over previously known solutions. For the line cutting version, we prove a key property that leads to a simple, constant factor approximation algorithm. For the ray cutting version, we prove it is possible to compute in almost linear time a cutting sequence that is an $O(\log^2 n)$ -factor approximation. No algorithms were previously known for the ray cutting version.

1 Introduction

About two decades ago Overmars and Welzl [6] have first considered the problem of cutting out a polygon in the cheapest possible way. The problem falls in the general area of stock cutting, where a given shape needs to be cut out from a parent piece of material, and it is defined as follows:



A $\dots\dots\dots$ is a line cut that does not cut through the interior of P and separates Q into a number of pieces, lying above and below the cut. A guillotine cut is an $\dots\dots\dots$ if it cuts along an edge of P . After a cut is made, Q is updated to that piece that still contains P . A $\dots\dots\dots$ is a sequence of cuts such that after the last cut in the sequence we have $P = Q$ (see Fig. 1). The cost of a cut is the length of the intersection of the cut with Q and the goal is to find a cutting sequence that minimizes the total cost.

From the definition of the problem it follows that P must be convex for a cutting sequence to exist. Overmars and Welzl [6] proved a number of properties for the case when both P and Q are convex polygons with n and m vertices, respectively, including: (i) There exists a finite optimal cutting sequence with at most $5n$ cuts, all touching P ; (ii) There are cases in which there is no optimal cutting sequence with all cuts along the edges of P and (iii) When only edge cuts are allowed, an optimal cutting sequence can be computed in $O(m + n^3)$ time by a dynamic programming algorithm. They further noted that when Q is not convex there are cases in which there is no optimal cutting sequence with all cuts touching P .

^{*} This research was partially supported by NSF grant CCF-0430366.

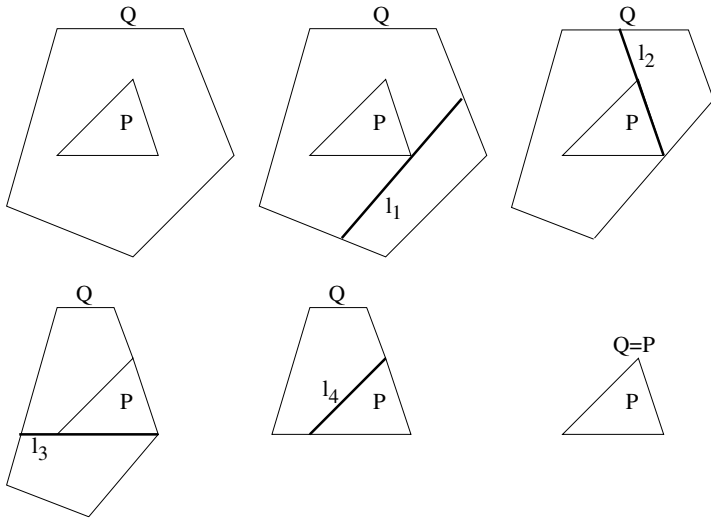


Fig. 1. A cutting sequence (bold lines) $\{l_1, l_2, l_3, l_4\}$ for cutting P out of Q

Overmars and Welzl [6] have also left a number of open problems including: (1) An algorithm for computing an optimal cutting sequence when Q is convex. Since the problem is not discrete, it might be possible that an optimal cutting sequence cannot be computed. (2) A proof of existence of a finite optimal cutting sequence when Q is not convex (regarding Q as an open object). (3) Algorithms to approximate an optimal cutting sequence. (4) The situation in which guillotine cuts are replaced by rays, where a ray runs from infinity to some point in Q . In this case, some non-convex polygons are \dots . These polygons have the property that every edge of the polygon must be extendible to a ray. The problem of computing or approximating optimal cutting sequences with rays has been left completely open.

About a decade later Bhadury and Chandrasekaran [1] have been able to answer the first open problem in a negative sense. Specifically, they showed that the problem has optimal solutions that lie in the algebraic extension of the field that the input data belongs to. Due to this algebraic nature of the problem, an approximation scheme is the best that one can achieve. They also provided an approximation scheme that, given an error range δ , is polynomial in δ and the encoding length of the input data in unary, and gives a cutting sequence $C(\delta)$ with total cost at most δ more than that of an optimal cutting sequence C^* .

For the fourth open problem, it has been proven in [2] that ray-cuttability can be tested in linear time. However, the problem itself (computing or approximating an optimal sequence of ray cuts) has been again left open.

Very recently, Dumitrescu [3] has proved that there exists an $O(\log n)$ -factor approximation algorithm, which runs in $O(mn + n \log n)$ time, for cutting out a convex polygon P with n vertices from a convex polygon Q with m vertices. Thus, he gives a first answer to the third open problem. He also raised the following two

interesting questions: (1) If Q is the minimum axis-aligned rectangle enclosing P , is an optimal edge-cutting sequence a constant factor approximation of an optimal cutting sequence? Answering this question in an affirmative sense would result in a constant-factor approximation algorithm for cutting P out of Q . (2) Is it possible to extend the results for guillotine cutting to ray cutting?

Some other related problems have been studied recently [5, 7]. In [5], Jaromczyk and Kowaluk consider cutting polyhedral shapes with a hot wire cutter and give an $O(n^5)$ time algorithm that constructs a cutting path, if one exists. In [7], Pach and Tardos consider the problem of separating a large family of subsets from a family of pairwise disjoint compact convex sets drawn on a sheet of glass.

Our Results. The main contributions of this paper are summarized below.

- We affirmatively answer the first open problem in [3], by proving there exists a sequence of edge cuts that is guaranteed to be a constant factor approximation of an optimal cutting sequence. We use this result to obtain a constant factor approximation algorithm for cutting P out of Q , when both P and Q are convex polygons, as follows. We first show how to cut out a triangle Q containing P of about the same size as P in $O((n+m)\log(n+m))$ time, with a sequence of cuts of total cost C only a constant factor more than the cost of an optimal cutting sequence C^* . This part of the algorithm is based on simple observations and a mapping to a (\dots, \dots) domain. Then, with an additional $O(n^3)$ time, we compute an optimal edge cutting sequence, resulting in an $O(n^3 + (n+m)\log(n+m))$ time, constant-factor approximation algorithm for cutting P out of Q . Alternatively, one can use a direct approach to obtain an $O(\log n)$ -factor approximation algorithm that requires only $O((n+m)\log(n+m))$ time, an improvement over the solution in [3] by nearly a linear factor in some cases.
- We address the ray-cutting version of the problem and give an approximation algorithm that computes in almost linear time a finite sequence of ray cuts that is guaranteed to be an $O(\log^2 n)$ -factor approximation of an optimal solution. We first show that when P is ray-cuttable and Q is the convex hull of P we can extend the approximation results for guillotine cutting to ray cutting, resulting in an $O(n \log n)$ time algorithm to compute an $O(\log^2 n)$ -factor approximation of the optimal ray cutting sequence. We next give an $O(\log n)$ -approximation algorithm for cutting out an n -vertex convex polygon P from an m -vertex convex polygon Q by a sequence of ray cuts. The running time of this algorithm is $O(n+m)$. Thus, when Q is convex, we answer the second question in [3] and the fourth question in [6] by combining the two algorithms above.

2 Cutting Out Polygons with Guillotine Cuts

Our approach for solving the problem resembles that in [3]. Specifically, the solution involves a separation phase and a carving phase. In the separation phase, three line cuts are used to obtain a triangle that encloses P and has perimeter

roughly the same as that of P . In the carving phase, all the cuts are along the edges of P .

2.1 Separation Phase

For a line l , let θ be the unique angle of l in the interval $I = [-90^\circ, 90^\circ]$. In [3], it has been proven that there exist two line cuts l_1 and l_2 with the property that (1) either the angle formed by l_1 and l_2 is such that $|\theta_1 - \theta_2| \in [20^\circ, 160^\circ]$, or l_1 and l_2 are almost parallel and tangent to P on opposite sides, (2) l_1 and l_2 are tangent to P and (3) $|l_1| + |l_2| = O(|C^*|)$, where C^* is an optimal cutting sequence. Two such cuts can be found in $O(nm + n \log n)$ time [3].

In this subsection, we show how to find in $O((n + m) \log(n + m))$ time a pair of cuts l_1 and l_2 that minimizes $|l_1| + |l_2|$ and satisfies the first two conditions above. Obviously, this implies the third condition, that is $|l_1| + |l_2| = O(C^*)$. The third cut is done as in [3], resulting in three line cuts of total length $O(|C^*|)$.

We reformulate the angle property above for cuts l_1 and l_2 by changing the definition of the angle θ of a cut l as follows. The angle of the line cut that is parallel to the x-axis and tangent to P from below is 0° . The angle increases gradually as the line cut rotates counter-clockwise along the boundary of P while being tangent to P . Thus, we have that $\theta \in [0^\circ, 360^\circ)$.

The problem now is to find two cuts l_1 and l_2 such that (1) l_1 and l_2 are tangent to P , (2) the angle between l_1 and l_2 is such that $|\theta_1 - \theta_2| \notin [0^\circ, 20^\circ)$ and (3) $|l_1| + |l_2|$ is minimized.

Let l be a line tangent to P and along the edge $v_{i-1}v_i$ of P , with $1 \leq i \leq n$ and $v_0 = v_n$, and consider rotating l around v_i until it overlaps with the edge $v_i v_{i+1}$ or it touches a vertex of Q . Then, in this angle interval $\theta \in [\theta_i^1, \theta_i^2]$, the length $|l| = l(\theta)$ of l is a convex function [1].

We can compute the functions $l(\theta)$ defining the cut length $|l|$ for all the $O(n + m)$ angle intervals in $O(n + m)$ time by rotating l along the boundary of P (similar to the ... technique [8]). The diagram of $l(\theta)$ is a continuous function that consists of $O(n + m)$ convex curves, and it has $O(n + m)$ local minima.

Our algorithm is as follows:

- 1: Let $\mathcal{M} = \{m_1, m_2, \dots, m_p\}$ be the list of local minima, where $p = O(m + n)$ is the number of local minima. Let the corresponding angles and cuts be $\mathcal{A} = \{a_1, a_2, \dots, a_p\}$ and $\mathcal{C} = \{c_1, c_2, \dots, c_p\}$, respectively. Assume the cuts are such that $a_1 < a_2 < \dots < a_p$.
- 2: Find the two cuts $l_1, l_2 \in \mathcal{C}$ of smallest length, with $|l_1| \leq |l_2|$. Let θ_1 and θ_2 be the angles of l_1 and l_2 , respectively.
- 3: Set $l = |l_1| + |l_2|$.
- 4: **if** $|\theta_1 - \theta_2| \in [0^\circ, 20^\circ)$ **then**
- 5: Set $l = \infty$.
- 6: Find all the local minima with angle in $(\theta_1 - 20^\circ, \theta_1 + 20^\circ)$, by a plane sweep from θ_1 . We obtain a set $\{m'_1, m'_2, \dots, m'_q\}$, where $q \leq p$, the corresponding angles are $\{a'_1, a'_2, \dots, a'_q\}$, with $a'_1 < a'_2 < \dots < a'_q$, and the corresponding cuts are $\{c'_1, c'_2, \dots, c'_q\}$.

- 7: **for** $i = 1$ to q **do**
- 8: Find the two cuts cb_1 and cb_2 with angles $a'_i - 20^\circ$ and $a'_i + 20^\circ$, respectively.
- 9: Find the smallest value m_0 among $\{m_1, m_2, \dots, m_p\} \cup \{cb_1, cb_2\} - \{m''_1, m''_2, \dots, m''_r\}$, where $r \leq p$ and $\{m''_1, m''_2, \dots, m''_r\}$ is the set of local minima in $\{m_1, m_2, \dots, m_p\}$ with angle in $(a'_i - 20^\circ, a'_i + 20^\circ)$.
- 10: **if** $m_0 + m'_i < l$ **then**
- 11: set $l_1 = c'_i$ and $l_2 = c_0$, where c_0 corresponds to the cut of length m_0 and c'_i corresponds to the cut of length m'_i .
- 12: set $l = |l_1| + |l_2|$.
- 13: **end if**
- 14: **end for**
- 15: **end if**

Theorem 1. $P \cap Q \neq \emptyset, P \subset Q, |l_1 - l_2| \notin [0^\circ, 20^\circ], l_{min} = |l_1| + |l_2|, O((n + m) \log(n + m))$

Proof. Use the algorithm above. The total running time is $O((n + m) \log(n + m))$ since finding the two smallest cuts in \mathcal{M} takes $O(n + m)$ time and the cost of the for loop is $O((n + m) \log(n + m))$ if we use plane sweep and a priority queue data structure. We next argue that $l_{min} = |l_1| + |l_2|$, as computed by the algorithm above, is of minimum length. Let θ_1 and θ_2 be the angles of the cuts l_1 and l_2 , respectively. Let θ_{min} be the angle corresponding to the cut of smallest length. We have $\theta_1 \in (\theta_{min} - 20^\circ, \theta_{min} + 20^\circ)$ and $\theta_2 \notin (\theta_1 - 20^\circ, \theta_1 + 20^\circ)$. Note that θ_1 and θ_2 could only be one of the $O(n + m)$ local minima except that θ_2 could also be at $\theta_1 - 20^\circ$ or $\theta_1 + 20^\circ$, if there is no local minima outside $(\theta_1 - 20^\circ, \theta_1 + 20^\circ)$ or all the local minima outside $(\theta_1 - 20^\circ, \theta_1 + 20^\circ)$ are larger than the cut length at $\theta_1 - 20^\circ$ or $\theta_1 + 20^\circ$. We check all possible pairs (l_1, l_2) during the plane sweep and maintain the smallest cut length for $|l_1| + |l_2|$ over all such pairs. \square

2.2 Carving Phase

In this section, we affirmatively answer the conjecture in [3]. Let C^* denote an optimal cutting sequence and let $|C^*|$ be its cost. Similarly, let C_e^* denote an optimal edge cutting sequence and let $|C_e^*|$ be its cost. Let $|P|$ and $|Q|$ denote the perimeter of P and Q , respectively.

Theorem 2. $P \cap Q \neq \emptyset, C_e^* \leq (2.5 + |Q|/|P|) |C^*|, P \subset Q$

Proof. We construct an edge-cutting sequence that is a $(2.5 + |Q|/|P|)$ -factor approximation of C^* as follows. For every optimal cut $C \in C^*$, in order, if C is an edge cut then we add it to the edge-cutting sequence. Otherwise, C is tangent to a vertex v of P and we add to the edge cutting sequence two cuts that are along the two edges of P incident at v (see Fig 2).

Consider an optimal cut C at a vertex v of P . Then, the endpoints of C are either on the boundary of Q or they are on some previous cuts of C^* . Let s_1

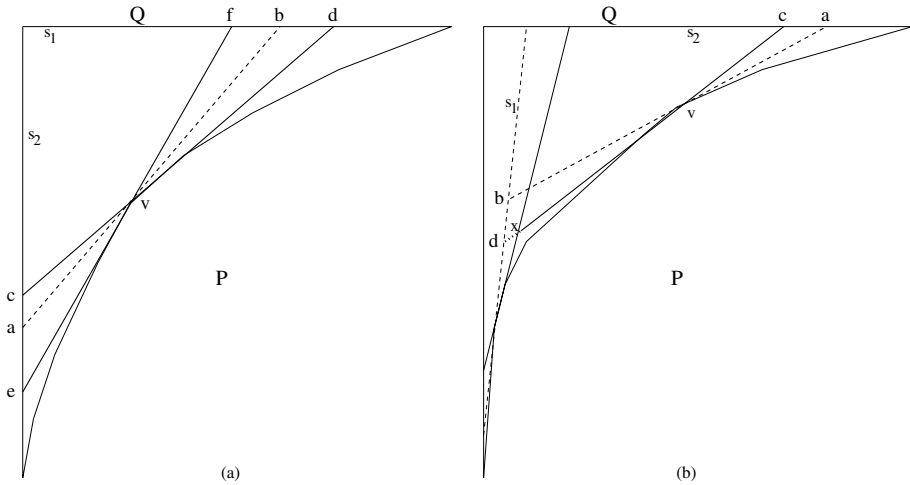


Fig. 2. Illustration of the extra cut cost, where dash lines are optimal cuts and continuous lines are edge cut. In (b), the current cuts are drawn with bold lines

and s_2 be the two line segments (of Q or C^*) on which the endpoints a and b of C lie. Then, C is opposite to an obtuse angle ($\geq 90^\circ$) in the triangle defined by C , s_1 and s_2 , as illustrated in Fig. 2. Let C_1 and C_2 be two edge cuts at v that replace the vertex cut at v . (The endpoints of C_1 and C_2 are on s_1 and s_2 . It will become clear below that C_1 and C_2 give upper bounds for the actual edge cuts we construct.) Then, v splits C_1 into two line segments, cv and vd , where cv is inside the obtuse triangle defined by C , s_1 and s_2 . Similarly, v splits C_2 into two line segments, ev and vf , where vf is inside the obtuse triangle defined by C , s_1 and s_2 . Clearly, $|cv| + |vf| < |ab|$ so at least one of $|cv|$ and $|vf|$ is smaller than half the length of the optimal cut C . Assuming $|cv| \leq |vf|$ we have $|cv| < |C|/2$. Then, the edge cuts C_1 and C_2 are done such that C_1 precedes C_2 and the cost of these two edge cuts is $|cv| + |ev| + |vd| < |C|/2 + |ev| + |vd|$. Thus, from now on, we focus on bounding the lengths of ev and vd . These lengths give upper bounds (see below) on the components of the two edge cuts performed along the edges incident to v , that replace the vertex cut at v . Let Q_e be the piece of Q that contains P after the two edge cuts are performed. Note that Q_e is included in the piece Q_v of Q that contains P after the optimal vertex cut C is performed. Since after a cut is performed the problem is divided into two independent subproblems, we need focus only on one of the two subproblems. Consider bounding the length of vd . We have two cases, depending whether s_1 is on the boundary of Q (Fig. 2 (a)) or it is part of some optimal cut preceding C in the optimal cutting sequence (Fig. 2 (b)). Let the cost of the optimal cut vb which goes through the vertex v of P be L_v , and let the cost of the corresponding edge cut in the edge cutting sequence we construct be L_e . The difference of L_e and L_v is the extra cost for the edge cut over the vertex cut. In the first case, bd is on the boundary of Q . Then, $L_e - L_v \leq E_Q$, where $E_Q = |bd|$. In the second

case, if d is on a vertex cut C' , note that the line segment vd must intersect an edge cut in the edge cutting sequence constructed so far (that edge cut is associated with C'). Let x be the intersection point. Thus, $|vx| \leq |vd|$ and the edge cut adds length $|vx|$ to the edge cutting sequence. The line segment bd is on a previous optimal cut in C^* and $L_e - L_v = |vx| - |vb| \leq |vd| - |vb| \leq |bd| = E_v$.

Since subproblems are independent, it is easy to see that for any two line segments l_1 and l_2 , which are on the boundary of Q or on an optimal cut and correspond to some values E_Q or E_v in the construction above, with $l_1 \neq l_2$, we have $l_1 \cap l_2 = \emptyset$, that is, they do not overlap. Then, the total extra cost is bounded by $|C^*| + |Q|$. Let C_e be the edge cutting sequence constructed. We have $|C_e| < |C^*|/2 + |C^*| + (|C^*| + |Q|) = 2.5|C^*| + |Q|$ and thus $|C_e|/|C^*| = 2.5 + |Q|/|C^*| < 2.5 + |Q|/|P|$. To end the proof we note that $|C_e^*| \leq |C_e|$. \square

An optimal edge cutting sequence C_e^* can be computed in $O(n^3 + m)$ time, where n and m are the number of vertices of P and Q , respectively. Alternatively, using an algorithm similar to that in [3], an $O(\log n)$ -factor approximation can be computed in $O(n)$ time, when $m = O(1)$ (we omit this proof due to space constraints). Combining Theorem 1 and Theorem 2 we have:

Theorem 3. $\dots P \dots Q \dots P \subset Q \dots O(1) \dots$
 $\dots P \dots Q \dots$
 $\dots O(n^3 + (n+m) \log(n+m)) \dots O(\log n) \dots$
 $\dots O((n+m) \log(n+m)) \dots$

3 Cutting Out Polygons with Ray Cuts

In this section we consider the ray cutting version of the problem: Given a convex polygonal piece of material Q with a ray cuttable polygon P drawn on it, cut P out of Q by a sequence of ray cuts in the cheapest possible way.

We first observe that the dynamic programming algorithm in [6] for computing an optimal edge cutting sequence does not work in the case of ray cuts. The dynamic programming algorithm is based on the following fact: if we consider a pair of cuts C_i and C_j then, after these two cuts are made, the cuts on the boundary of P between C_i and C_j , in clockwise order, are independent of the cuts between C_i and C_j taken in counter-clockwise order. However, in general this is not true for a ray cutting sequence. Consider Fig. 3 and suppose we have two cuts ab and bc followed by other two cuts dg and ef . Then, the length of the cut dg , which is on the boundary of P between ab and bc in clockwise order, depends on whether ef is made before or after dg , where the cut ef is between the cuts ab and bc in counter-clockwise order.

Our solution for approximating an optimal ray cutting sequence has two key steps. In the first step, we assume that Q is the convex hull of P and show how to cut out a pocket, where a pocket is defined by an edge e of Q , $e \notin P$, and the boundary of P interior to Q and between the endpoints of e (see Fig. 3). In the second step, we show how to cut out a convex polygon by a sequence of ray cuts that is a good approximation of an optimal ray cutting sequence. As in the

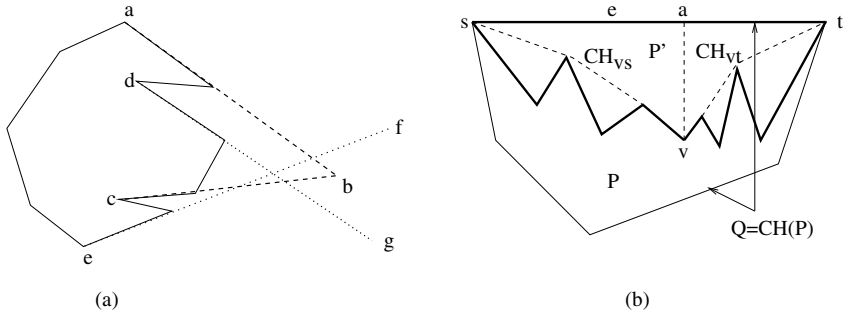


Fig. 3. (a) Illustrating that dynamic programming may fail. (b) A pocket P' of P

line cutting problem, this step has two phases: a separation phase and a carving phase. We mention that we do not address the existence of a finite optimal ray cutting sequence. This remains an open problem.

3.1 Cutting Out Pockets by Rays

Let $CH(P)$ be the convex hull of P . Let s and t be the end vertices of an edge $e \in CH(P)$ defining a pocket P' of P . For a vertex $v \in P$ that is inside the pocket P' , let CH_{vs} (resp. CH_{vt}) be the portion of the boundary of the convex hull of the vertices of P' between v and s (resp., between v and t) that lies in the pocket P' (see Fig. 3). Note that the shortest paths from v to s and t are the same as CH_{vs} and CH_{vt} and that the two subproblems of cutting out the pockets of P defined by CH_{vs} and CH_{vt} are independent (when CH_{vs} and CH_{vt} are seen as the boundary of Q for the subproblems).

We use the simplified edge cutting algorithm for the carving phase in Section 2.2, with some modifications, to cut out CH_{vs} and CH_{vt} from the pocket of v . We first make a ray cut va , where va is the shortest straight line segment from v to the segment st that has empty intersection with P . Such a ray cut always exists since the polygon P is ray cuttable, and it can be found in constant time. After the cut va , the ray cuts we perform for CH_{vs} or CH_{vt} originate on st and do not cross va . Consider cutting out the polygonal line CH_{vs} (we can assume Q is the triangle defined by sa , av and vs and P is $CH_{vs} \cup \{vs\}$). Then, every edge cut in an edge cutting sequence of the boundary of CH_{vs} that intersects with va corresponds to a ray with the property above.

Theorem 4. $|Q| \dots \dots \dots |P| \dots \dots \dots C_r \dots \dots \dots$
 $\dots O(|P| \log^2 n) \dots \dots \dots O(n \log n) \dots \dots \dots C_r \dots \dots \dots$
 $|C_r| = O(|C^*| \log^2 n), \dots \dots \dots C^* \dots \dots \dots$

Proof. The second part of the theorem is obvious, since $|P| = O(|C^*|)$. We only sketch the proof for the first part. Consider a pocket of P , defined by a segment st as above. Let P' be the portion of P in this pocket. To compute C_r we first find the middle vertex v of the pocket, compute the shortest paths CH_{vs} and CH_{vt} from v to s and t , and make the ray cut av . We cut out CH_{vs} and CH_{vt} using edge cuts. Note that the polygon defined by CH_{vs} , CH_{vt} and

the line segment st is cut out of P' , and P' is separated into two parts: a left part, P'_l , and a right part, P'_r . Let v_l be the middle vertex of P'_l and let v_r be the middle vertex of P'_r . Next, we cut out $CH_{v_l s}, CH_{v_l t}, CH_{v_r s}$ and $CH_{v_r t}$ using edge cuts, and proceed recursively until P' is cut out. It can be easily shown that at any level of recursion the corresponding cuts av and the portions of Q and P' for various subproblems have about the same size. There are $O(\log n)$ recursion levels and at each level the cost of the cut is $O(|P'| \log n)$ if we use the edge cutting algorithm in the carving phase of Theorem 3. Thus, the total cost is $O(|P'| \log^2 n)$. We can compute the shortest paths CH_{vs} and CH_{vt} in linear time [4]. The time to cut CH_{vs} and CH_{vt} is $O(n_{st})$, where n_{st} is the number of vertices of P' . Then, over all pockets of P the total time is $O(n \log n)$. \square

3.2 Cutting Out Convex Polygons by Rays

As in the line cutting problem, we use a two phase algorithm for cutting out a convex polygon P from a convex polygon Q : a separation phase and a carving phase. In the separation phase, three ray cuts are made. After those cuts, P is enclosed by a triangle Q' such that $diam(Q')/diam(P) = O(1)$, where $diam(P)$ (resp., $diam(Q')$) denotes the diameter of P (resp., Q'). In the carving phase we cut P from Q' by a sequence of cuts along the edges of P .

Separation Phase. Let ∂P (resp. ∂Q) denote the boundary of P (resp. Q). Let n be the number of vertices of P and let m be the number of vertices of Q .

Lemma 1. $O(n + m)$ ∂P ∂Q $O(m \log n)$

Let \overline{pq} be the segment that gives the closest distance, where p is a vertex of P and q is on some edge $e \in \partial Q$. Let l_q be the supporting line of e . Note that the angle between \overline{pq} and l_q is a right angle (see Fig. 4). Let l_s be the semi-line originating from q and going through p , splitting P into two sides (left side and right side). Consider two ray cuts bj and bo tangent to P at vertices d and e respectively, and ending at the same point b on l_s , where j and o are the originating points of the two ray cuts on l_q . Assume that point b moves along l_s and let $C_b = |bj| + |bo|$. Then, we can compute the minimum value C_{bmin} of C_b in $O(n)$ time.

Lemma 2. C_{bmin} C^* $|C_{bmin}| \leq 9 \cdot |C^*|$

Proof. Let l'_q be the line tangent to P and parallel to l_q , such that P is between l_q and l'_q , and refer to Fig. 4. The line l'_q is also orthogonal to l_s . Let a, h and i be the intersection points of l'_q with l_s, bj and bo , respectively. Let $D = diam(P)$, let $|ah| = x$ and let $D' = |hi|$. Then, $|ai| = D' - x$ and $D' \leq D$. Let $|aq| = S, |ab| = y, l_1 = |bh|, l_2 = |hj|, l_3 = |bi|$ and $l_4 = |io|$. We then have: $\frac{l_1+l_2}{S+y} = \frac{l_1}{y}, l_1 + l_2 = (S + y)\frac{l_1}{y}, \frac{l_3+l_4}{S+y} = \frac{l_3}{y}, l_3 + l_4 = (S + y)\frac{l_3}{y}$ and $C_{bmin} \leq C_b = l_1 + l_2 + l_3 + l_4 = \frac{S+y}{y}(l_1+l_3) = (\frac{S}{y}+1)(l_1+l_3) \leq (\frac{S}{y}+1)(2y+D') \leq (\frac{S}{y}+1)(2y+D)$, since $l_1 \leq x+y, l_3 \leq y + D' - x$ and $D' \leq D$. Let $f(y) = (S/y + 1)(2y + D)$. $f(y)$ is convex and

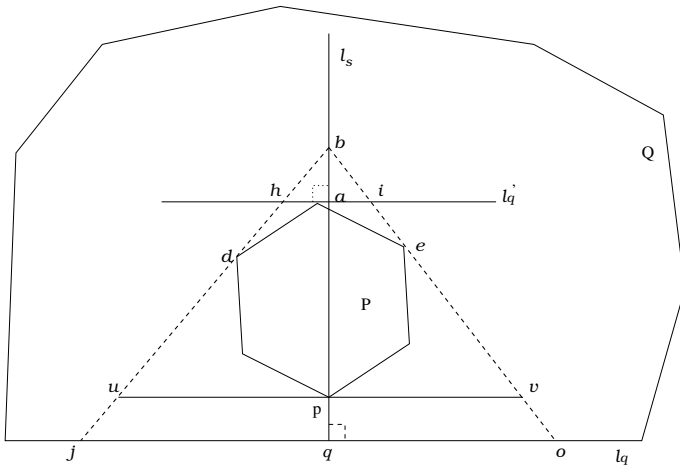


Fig. 4. Illustration of the ray cut construction

to minimize it we set $\partial f / \partial y = 0$ and get $y = \sqrt{SD/2}$. We then obtain $C_{bmin} \leq (\frac{S}{\sqrt{SD/2}} + 1)(2\sqrt{SD/2} + D) = (\sqrt{2S/D} + 1)(\sqrt{2SD} + D)$. Since $S = |aq| = |ap| + |pq|$, $|pq| \leq |C^*|$ and $|ap| \leq D \leq |C^*|$, we have $S \leq 2|C^*|$. Then, $C_{bmin} \leq (\sqrt{4|C^*|/D} + 1)(\sqrt{4D|C^*|} + D) = 4|C^*| + 4\sqrt{|C^*|D} + D \leq 9|C^*|$, as $D \leq |C^*|$. \square

After the cuts bj and bo for C_{bmin} are made, to cut out P we cut along the line uv , where uv is the cut through p and parallel to l_q , and u and v are the two intersection points of this line with bj and bo . Obviously, $|uv| \leq |bj| + |bo| \leq 9 \cdot |C^*|$. Thus, the total cost for the separating phase is $|uv| + |bj| + |bo| \leq 18 \cdot |C^*|$, which is a constant factor from an optimal solution. We then obtain:

Theorem 5. $C \leq 18 \cdot |C^*|$, $Q' \subset P \subset Q$, $|Q'|/|P| = O(1)$, $O(m + n)$.

Carving Phase. We first give a number of properties of an optimal ray cutting sequence, when Q is a convex polygon. Then, by combining the results in this subsection with those in Section 3.1 we obtain our main approximation results.

Lemma 3. ∂P .

Proof. Suppose an optimal ray cut r_1 does not end on ∂P or on another ray cut. There are two possible cases. **Case 1.** The ray r_1 does not intersect with other ray cuts (see Fig. 5 (a)). Obviously we can discard r_1 , since r_1 does not cut off any part of Q and it has no contribution in cutting P out of Q . **Case 2.** The ray r_1 intersects with other optimal ray cuts but its endpoint v_1 does not lie on any other optimal ray cut (see Fig 5 (b)). Let r_2 be the last ray intersected

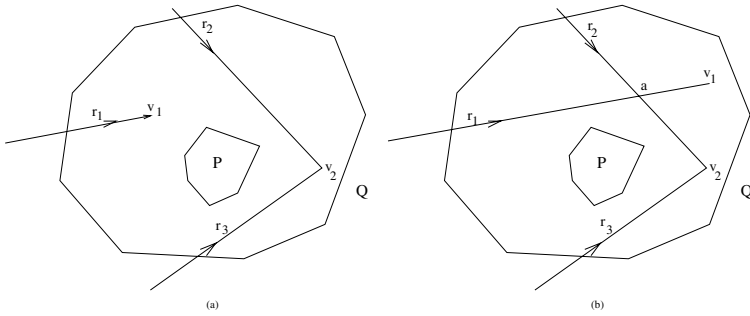


Fig. 5. Illustrating that a ray cut r_1 must end on ∂P or on another ray cut

by r_1 and let a be the intersection point of r_1 and r_2 . Then, we can discard the line segment av_1 using a similar argument as in **Case 1**.

Lemma 4. *..... P*

Proof. We make the proof by contradiction. Suppose that there is an optimal ray cut r_1 that does not touch P . From Lemma 5, we know that r_1 ends on another optimal ray cut (denote it by r_2). Let $v_1 = r_1 \cap r_2$. Consider another optimal ray cut r_4 that ends on r_1 (see Fig. 6). We can move r_1 parallel to itself either towards P or away from P and at the same time keep the end point v_1 on r_2 . For the ray cut r_4 , which ends on r_1 at v_3 , we let v_3 move with r_1 . The function that gives the total change in length for $|r_1| + |r_4|$ is a linear function. Then, this situation is the same as in the proof that an optimal line cut must touch P [1] and it follows that r_1 either is not necessary or it must touch P . \square

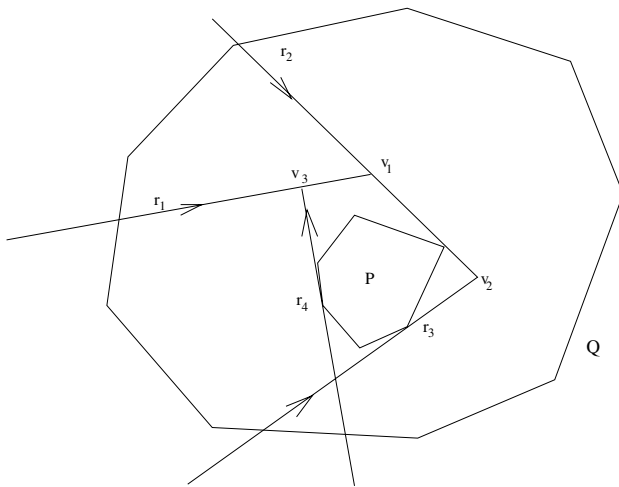


Fig. 6. Illustrating that the ray cut r_1 either touches P or it is not needed

Lemma 5. *Let P be a ray-cuttable polygon with n vertices and m edges. Then, $CH(P)$ can be cut out from Q in $O(n^3 + m)$ time and with a cost of $O(\log^2 n)$.*

Proof. We first use the separation algorithm in Section 3.2 to obtain a triangle Q' containing P such that all edges of Q' touch P and $diam(Q') = O(diam(P))$. This takes $O(m + n)$ time. Note that from the way we upper bound C_{bmin} in Section 3.2, it also follows that $|C_{bmin}| \leq 9 \cdot |C^*|$, where $|C^*|$ is the optimal cost of cutting out a ray-cuttable polygon P from a convex polygon Q . This is true since in the proof of Lemma 2 we only use $|C^*|$ to upper bound the diameter of P and the closest distance from P to Q . Then, the cost to cut out Q' when P is a ray-cuttable polygon is upper bounded by $18 \cdot |C^*|$, and thus by $O(|C^*|)$.

In the carving phase, we compute an optimal edge cutting sequence to cut out $CH(P)$ from Q' , where $CH(P)$ is the convex hull of P . The optimal edge cutting sequence C_e^* has cost bounded by $O(|P| \log n)$ and can be found in $O(n^3)$ time (see Section 2.2). We then cut out P from $CH(P)$ using the algorithm for cutting out pockets in Section 3.1. Adding up, the total cutting cost is $O(|C^*|) + O(|P| \log n) + O(|P| \log^2 n) = O(|C^*| + |P| \log^2 n) = O(|C^*| \log^2 n)$ and the running time is $O((n + m) + n^3 + n \log n) = O(n^3 + m)$. \square

Theorem 6. *Let P be a ray-cuttable polygon with n vertices and m edges. Then, $CH(P)$ can be cut out from Q in $O(m + n \log n)$ time and with a cost of $O(\log^2 n)$.*

Proof. Same as the proof of Lemma 5, except that in the carving phase, instead of an optimal edge cutting sequence C_e^* , we compute in $O(n)$ time an approximate edge cutting sequence C_e that has cost $O(|P| \log n)$, as in the carving phase for Theorem 3 (see the end of Section 2.2, paragraph before Theorem 3). Then, the total running time is $O((m + n) + n + n \log n) = O(m + n \log n)$. \square

References

1. Bhadury, J. and Chandrasekaran, R.: Stock cutting to minimize cutting length. *European Journal of Operational Research*. **88** (1996) 69–87.
2. Demaine, E.D., Demaine, M.L. and Kaplan, C.S.: Polygons cuttable by a circular saw. *Computational Geometry: Theory and Applications*. **20** (2001) 69–84.
3. Dumitrescu, A.: An approximation algorithm for cutting out convex polygons. *Procs. of the 14-th ACM-SIAM Symposium on Discrete Algorithms*. (2003) 823–827.
4. Guibas, L., Hershberger, J., Leven, D., Sharir, M. and Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*. **2(2)** (1987) 209–233.
5. Jaromczyk, J.W. and Kowaluk, M: Sets of lines and cutting out polyhedral objects. *Computational Geometry*. **25** (2003) 67–95.
6. Overmars, M.H. and Welzl, E.: The complexity of cutting paper. *Procs. of the 1st Annual ACM Symposium on Computational Geometry*. (1985) 316–321.
7. Pach, J. and Tardos, G.: Cutting Glass. *Disc. & Comput. Geom.* **24** (2000) 481–495.
8. Toussaint, G.T.: Solving geometric problems with the ‘rotating calipers’. *Procs. MELECON, Athens, Greece, 1983*.

Advantages of Backward Searching — Efficient Secondary Memory and Distributed Implementation of Compressed Suffix Arrays

Veli Mäkinen¹, Gonzalo Navarro^{2,*}, and Kunihiro Sadakane^{3,**}

¹ Dept. of Computer Science, Univ. of Helsinki, Finland
vmakinen@cs.helsinki.fi

² Center for Web Research, Dept. of Computer Science, Univ. of Chile, Chile
gnavarro@dcc.uchile.cl

³ Dept. of Computer Science and Communication Engineering, Kyushu Univ., Japan
sada@csce.kyushu-u.ac.jp

Abstract. One of the most relevant succinct suffix array proposals in the literature is the Compressed Suffix Array (CSA) of Sadakane [ISAAC 2000]. The CSA needs $n(H_0 + O(\log \log \sigma))$ bits of space, where n is the text size, σ is the alphabet size, and H_0 the zero-order entropy of the text. The number of occurrences of a pattern of length m can be computed in $O(m \log n)$ time. Most notably, the CSA does not need the text separately available to operate. The CSA simulates a binary search over the suffix array, where the query is compared against text substrings. These are extracted from the same CSA by following irregular access patterns over the structure. Sadakane [SODA 2002] has proposed using *backward searching* on the CSA in similar fashion as the FM-index of Ferragina and Manzini [FOCS 2000]. He has shown that the CSA can be searched in $O(m)$ time whenever $\sigma = O(\text{polylog}(n))$.

In this paper we consider some other consequences of backward searching applied to CSA. The most remarkable one is that we do not need, unlike all previous proposals, any complicated sub-linear structures based on the four-Russians technique (such as constant time *rank* and *select* queries on bit arrays). We show that sampling and compression are enough to achieve $O(m \log n)$ query time using less space than the original structure. It is also possible to trade structure space for search time. Furthermore, the regular access pattern of backward searching permits an efficient secondary memory implementation, so that the search can be done with $O(m \log_B n)$ disk accesses, being B the disk block size. Finally, it permits a distributed implementation with optimal speedup and negligible communication effort.

* Funded by Millennium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile.

** Partially funded by the Grant-in-Aid of the Ministry of Education, Science, Sports and Culture of Japan.

1 Introduction

The classical problem in string matching is to determine the occurrences of a short pattern $P = p_1p_2 \dots p_m$ in a large text $T = t_1t_2 \dots t_n$. Text and pattern are sequences of characters over an alphabet Σ of size σ . Usually the same text is queried several times with different patterns, and therefore it is worthwhile to preprocess the text in order to speed up the searches. Preprocessing builds an index structure for the text.

To allow fast searches for patterns of any size, the index must allow access to all suffixes of the text (the i th suffix of T is $t_it_{i+1} \dots t_n$). These kind of indexes are called *suffix indexes*. Optimal query time, which is $O(m)$ as every character of P must be examined, can be achieved by using the *suffix tree* [25, 12, 23] as the index.

The suffix tree takes much more memory than the text. In general, it takes $O(n \log n)$ bits, while the text takes $n \log \sigma$ bits¹. A smaller constant factor is achieved by the *suffix array* [10]. Still, the space complexity does not change. Moreover, the searches take $O(m \log n)$ time with the suffix array (this can be improved to $O(m + \log n)$ using twice the original amount of space [10]).

The large space requirement of full-text indexes has raised the interest on indexes that occupy the same amount of space as the text itself, or even less. For example, the Compressed Suffix Array (CSA) of Sadakane [19] takes in practice the same amount of space as the text compressed with a zero-order model. Moreover, the CSA does not need the text at all, since the text is included in the index. Existence and counting queries on the CSA take $O(m \log n)$ time.

There are also other so-called *space-optimal* full-text indexes that achieve good tradeoffs between search time and space complexity [3, 9, 7, 22, 5, 14, 18, 16, 4]. Most of these are *self-indexes* as they take less space than the text itself, and also *self-indexes* as they contain enough information to reproduce the text: A self-index does not need the text to operate.

Recently, several space-optimal self-indexes have been proposed [5, 6, 4], whose space requirement depends on the k -th order empirical entropy with constant factor one (except for the sub-linear parts). These indexes achieve good query performances in theory, but they are complex to implement as such.

2 Summary of Results

In this paper, we concentrate on simplifying and generalizing earlier work on succinct self-indexes. We build on the Sadakane’s CSA [19], which we briefly review in the following.

The CSA searches the pattern by simulating a binary search over the suffix array. The search string must be compared against some text substring at each step of the binary search. Since the text is not stored, the CSA must be traversed

¹ By log we mean \log_2 in this paper.

in irregular access patterns to extract each necessary text substring. This makes it unappealing e.g. for a secondary memory implementation.

Sadakane [20] has proposed using rank and select on the CSA in similar fashion as the FM-index of Ferragina and Manzini [3]. Sadakane has shown that the CSA can be searched in $O(m)$ time whenever $\sigma = O(\text{polylog}(n))$. The CSA scales much better than the FM-index as the alphabet size grows.

Backward searching has also some other consequences than improved search time, as we will show in this paper. We exploit the fact that the access pattern becomes much more regular. The most important consequence of this is that a simpler implementation of the CSA is possible: All previous proposals heavily rely on sublinear structures based on the so-called four-Russians technique [1] to support constant time rank and select queries on bit arrays [8, 13, 2] ($\text{rank}(i)$ to find out how many bits are set before position i , and $\text{select}(j)$ to find out the position of the j th bit from the beginning). We show that these structures are not needed for an efficient solution, but rather one can do with sampling and traditional compression. This is a distinguishing feature of our proposal. The absence of four-Russians tricks makes our index usable on weaker machine models and also makes it easier to implement.

Under this simpler implementation scenario, we are able to retain the original $O(m \log n)$ search time, and improve the original $n(H_0 + O(\log \log \sigma))$ space to $n(H_0 + \varepsilon)(1 + o(1))$, for any $\varepsilon > 0$. The search time can be reduced gradually, to $O(\lceil m/\ell \rceil \log n)$ time, up to $O(m \log \sigma + \log n)$. The price is that the space requirement increases to $n(\sum_{i=0}^{\ell-1} H_i + \varepsilon)(1 + o(1))$, being H_i the order- i empirical entropy of T . We also give an alternative implementation where the space requirement depends on H_k , for any k . Furthermore, the CSA becomes amenable of a secondary memory implementation. If we call B the disk block size, then we can search the CSA in secondary memory using $O(m \log_B n)$ disk accesses. Finally, we show that the structure permits a distributed implementation with optimal speedup and negligible communication effort. Table 1 compares the original and the new CSA.

Table 1. Space and time complexities of the original and new CSA implementations

	Original CSA	Our CSA, version 1	Our CSA, version 2
Space (bits)	$n(H_0 + O(\log \log \sigma))$	$n(\sum_{i=0}^{\ell-1} H_i + \varepsilon)$	$2n(H_k(\log \sigma + \log \log n) + \varepsilon)$
Search time	$O(m \log n)$	$O(\lceil m/\ell \rceil \log n)$	$O(m \log n)$
Disk search time	$O(m \log n)$	$O(\lceil m/\ell \rceil \log_B n)$	$O(m \log_B n)$
Remote messages	$m \log n$	$\lceil m/\ell \rceil$	m

Our solution takes about the same amount of space as Sadakane's improvement in [20]. Our search time is worse by a $\log n$ factor. However, our structure works on more general alphabets; we only assume that $\sigma = o(n/\log n)$, as Sadakane's improvement in [20] assumes that $\sigma = O(\text{polylog}(n))$. The space-optimal solutions [5, 6, 4] have also similar restrictions on the alphabet size.

3 The Compressed Suffix Array (CSA) Structure

Let us first describe the basic suffix array data structure. Given a text $T = t_1t_2 \dots t_n$, we consider the n text suffixes $t_jt_{j+1} \dots t_n$, so that the j -th suffix of T is $t_jt_{j+1} \dots t_n$. We assume that a special endmarker “\$” has been appended to T , such that the endmarker is smaller than any other text character. The suffix array \mathcal{A} of T is the set of suffixes $1 \dots n$, arranged in lexicographic order. That is, the $\mathcal{A}[i]$ -th suffix is lexicographically smaller than the $\mathcal{A}[i+1]$ -th suffix of T for all $1 \leq i < n$.

Given the suffix array, the search for the occurrences of the pattern $P = p_1p_2 \dots p_m$ is trivial. The occurrences form an interval $[sp, ep]$ in \mathcal{A} such that suffixes $t_{\mathcal{A}[i]}t_{\mathcal{A}[i]+1} \dots t_n$, $sp \leq i \leq ep$, contain the pattern as a prefix. This interval can be searched for using two binary searches in time $O(m \log n)$.

The compressed suffix array (CSA) structure of Sadakane [19] is based on that of Grossi and Vitter [7]. In the CSA, the suffix array $\mathcal{A}[1 \dots n]$ is represented by a sequence of numbers $\Psi(i)$, such that $\mathcal{A}[\Psi(i)] = \mathcal{A}[i] + 1$. Furthermore, the sequence is differentially encoded, $\Psi(i) - \Psi(i - 1)$. If there is a suffix $\mathcal{A}[j \dots j + \ell]$, that is $\mathcal{A}[j \dots j + \ell] = \mathcal{A}[i \dots i + \ell] + 1$, then $\Psi(i \dots i + \ell) = j \dots j + \ell$, and $\Psi(i') - \Psi(i' - 1) = 1$ for $i < i' \leq i + \ell$. Hence the differential array is encoded with a method that favors small numbers and permits constant time access to Ψ . Note in particular that Ψ values are increasing in the areas of \mathcal{A} where the suffixes start with the same character a , because $ax < ay$ iff $x < y$.

Additionally, the CSA stores an array $C[1 \dots \sigma]$, such that $C[c]$ is the number of occurrences of characters $\{\$, 1, \dots, c - 1\}$ in the text T . Notice that all the suffixes $\mathcal{A}[C[c] + 1] \dots \mathcal{A}[C[c + 1]]$ start with character c . The text is discarded.

A binary search over \mathcal{A} is simulated by extracting from the CSA strings of the form $t_{\mathcal{A}[i]}t_{\mathcal{A}[i]+1}t_{\mathcal{A}[i]+2} \dots$ for any index i required by the binary search. The first character $t_{\mathcal{A}[i]}$ is easy to obtain because all the first characters of suffixes appear in order when pointed from \mathcal{A} , so $t_{\mathcal{A}[i]}$ is the character c such that $C[c] < i \leq C[c+1]$. This is found in constant time by using small additional structures based on the four-Russians technique [8, 13, 2]. Once the first character is obtained, we move to $i' \leftarrow \Psi(i)$ and go on with $t_{\mathcal{A}[i']} = t_{\mathcal{A}[i]+1}$. We continue until the result of the lexicographical comparison against the pattern P is clear. The overall search complexity is the same as with the original suffix array, $O(m \log n)$.

Note that each string comparison may require accessing up to m arbitrary cells in the Ψ array (see Fig. 1). Hence using the CSA in secondary memory is not attractive because of the scattered access pattern. Also, a complex part in the implementation of the CSA is the compression of the Ψ array, since it must support constant time direct access at any position. This is achieved in [19] by using four-Russians techniques, in $n(H_0 + O(\log \log \sigma))$ bits of space.

Notice that the above search only solves counting and reporting queries: We find the interval of the suffix array that would contain suffixes of the text matching the pattern. The pointers to suffixes are not stored explicitly, and hence we cannot report the occurrences or show the text context around them. The solution is to sample suffixes $1, \log n, \dots$, and use the Ψ function to retrieve the unsampled ones [19]. We will only consider counting queries in the sequel, since we can use the sampling technique as is to report occurrences.

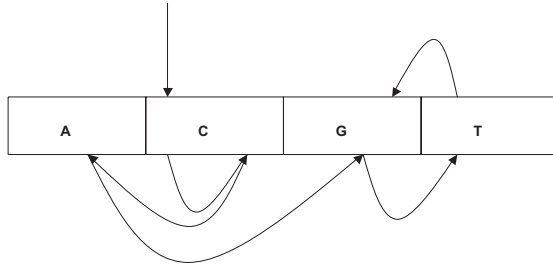


Fig. 1. One step of binary search for pattern $P = CCAGTA$. The blocks correspond to the areas of suffix array whose suffixes start with the corresponding letter. The straight arrow on top indicates the suffix the pattern is compared against. Other arrows indicate the extraction of the prefix of the compared suffix. The extraction ends at letter G , and hence the suffix does not correspond to an occurrence, and the search is continued to the left of the current point

4 Backward Search on CSA

Sadakane [20] has proposed using Ψ on the CSA. Let us review how this search proceeds. We use the notation $R(X)$, for a string X , to denote the range of suffix array positions corresponding to suffixes that start with X . The search goal is therefore to determine $R(P)$. We start by computing $R(p_m)$ simply as $R(p_m) = [C[p_m] + 1, C[p_m + 1]]$. Now, in the general case, given $R(P[i + 1 \dots m])$, it turns out that $R(P[i \dots m])$ consists exactly of the suffix array positions in $R(p_i)$ containing values j such that $j + 1$ appears in suffix array positions in $R(P[i + 1 \dots m])$. That is, the occurrences of $P[i \dots m]$ are the occurrences of p_i followed by occurrences of $P[i + 1 \dots m]$. Since $\mathcal{A}[\Psi(i)] = \mathcal{A}[i] + 1$, it turns out that

$$x \in R(P[i \dots m]) \Leftrightarrow x \in R(p_i) \wedge \Psi(x) \in R(P[i + 1 \dots m])$$

Now, Ψ can be accessed in constant time, and its values are increasing inside $R(p_i)$. Hence, the set of suffix array positions x such that $\Psi(x)$ is inside some range forms a continuous range of positions and can be binary searched inside $R(p_i)$, at $O(\log n)$ cost. Therefore, by repeating this process m times we find $R(P)$ in $O(m \log n)$ time.

Fig. 2 gives the pseudocode of the algorithm, and Fig. 3 illustrates.

Note that the backward search (as explained here) does not improve the original CSA search cost. However, it is interesting that the backward search does not use the text at all, while the original CSA search algorithm is based on the concept of extracting text strings to compare them against P . These string extractions make the access pattern to array Ψ irregular and non-local.

In the backward search algorithm, the accesses to Ψ always follow the same pattern: binary search inside $R(c)$, for some character c . In the next sections we study different ways to take advantage of this feature. This is where our exposition differs from [20].

Algorithm *BackwardCSA*(P, C, Ψ):
 $left_{m+1} := 1; right_{m+1} := n;$
for $i := m$ **downto** 1 **do begin**
 $left_i = \min\{j \in [C[p_i] + 1, C[p_i + 1]], \Psi(j) \in [left_{i+1}, right_{i+1}]\};$
 $right_i = \max\{j \in [C[p_i] + 1, C[p_i + 1]], \Psi(j) \in [left_{i+1}, right_{i+1}]\};$
if $left_i > right_i$ **return** “no occurrences found”;
return “ $right_1 - left_1 + 1$ occurrences found”

Fig. 2. Backward search algorithm over the CSA. Functions “min” and “max” stand for binary searches

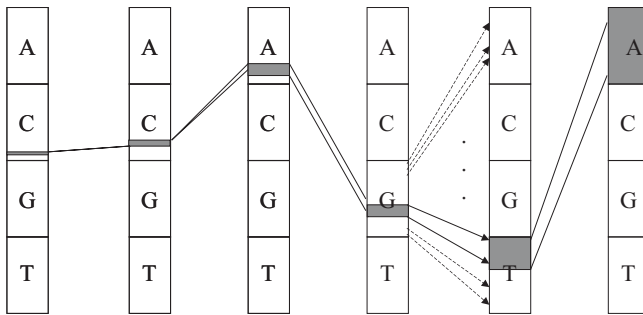


Fig. 3. Searching for pattern $P = CCAGTA$ backwards (right-to-left). The situation after reading each character is plotted. The gray-shaded regions indicate the interval of the suffix array that contain the current pattern suffix. The computation of the new interval is illustrated in the second step (starting from right). The Ψ values from the block of letter G point to consecutive positions in the suffix array. Hence it is easy to binary search the top-most and bottom-most pointers that are included in the previous interval

5 Improved Search Complexity

A first improvement due to the regular access pattern is the possibility of reducing the search complexity from $O(m \log n)$ to $O(m \log \sigma + \log n)$. Albeit in [20] they obtain $O(m)$ search time, the more modest improvement we obtain here does not need any four-Russians technique.

The idea is that we can extend the C array so as to work over strings of length ℓ (ℓ -grams) rather than over single characters. Given ℓ -gram x , $C[x]$ is the number of text ℓ -grams that are lexicographically smaller than x . The final $\ell - 1$ suffixes of length less than ℓ are accounted as ℓ -grams by appending them as many “\$” characters as necessary.

With this C array, we can search for pattern P of length m in $O(\lceil m/\ell \rceil \log n)$ time as follows. We first assume that m is a multiple of ℓ . Let us write $P =$

$G_1 G_2 \dots G_{m/\ell}$, where all G_i are all of length ℓ . We start by directly obtaining $R(G_{m/\ell}) = [C[G_{m/\ell}] + 1, C[\text{next}(G_{m/\ell})]]$, where $\text{next}(x)$ is the string of length $|x|$ that lexicographically follows x (if no such string exists, then assume $C[\text{next}(G_{m/\ell})] = n$). Once this is known, we binary search in $R(G_{m/\ell-1})$ the subinterval that points inside $R(G_{m/\ell})$. This becomes $R(G_{m/\ell-1} G_{m/\ell})$. The search continues until we obtain $R(P)$. The number of steps performed is m/ℓ , each being a binary search of cost $O(\log n)$.

Let us consider now the case where ℓ does not divide m . We extend P so that its length is a multiple of ℓ . Let $e = m - (m \bmod \ell)$. Then we build two patterns out of P . The first is P_l , used to obtain the left limit of $R(P)$. P_l is the lexicographically smallest ℓ -gram that is not smaller than P , $P_l = P\e , that is, P followed by e occurrences of character “\$”. The second is P_r , used to obtain the right limit of $R(P)$. P_r is the smallest ℓ -gram that is lexicographically larger than any string prefixed by P , $P_r = \text{next}(P)\e . Hence, we search for P_l and P_r to obtain $R(P_l) = [sp_l, ep_l]$ and $R(P_r) = [sp_r, ep_r]$. Then, $R(P) = [sp_l, sp_r - 1]$.

Note that $\text{next}(x)$ is not defined if x is the largest string of its length. In this case we do not need to search for P_r , as we use $sp_r = n + 1$.

We have obtained $O(\lceil m/\ell \rceil \log n)$ search time, at the cost of a C table with σ^ℓ entries. If we require that C can be stored in n bits, then $\sigma^\ell \log n = n$, that is, $\ell = \log_\sigma n - \log_\sigma \log n$. The search complexity becomes $O(\lceil m/\log_\sigma n \rceil \log n) = O(m \log \sigma + \log n)$ as promised.

Moreover, we can reduce the C space complexity to $O(n/\log^t n)$ for any constant t . The condition $\sigma^\ell \log n = n/\log^t n$ translates to $\ell = \log_\sigma n - (t + 1) \log_\sigma \log n$, and the search cost remains $O(m \log \sigma + \log n)$.

Notice that we cannot use the original Ψ function anymore to find the subintervals, since we read ℓ characters at a time. Instead, we need to store values $\Psi^\ell[i] = \Psi[\Psi[\dots \Psi[i]] \dots]$, where Ψ function is recursively applied ℓ times. Next section considers how to represent the Ψ^ℓ values succinctly.

6 A Simpler and Smaller CSA Structure

One of the difficulties in implementing the CSA is to provide constant time access to array Ψ (or Ψ^ℓ using the search procedure from previous section). This is obtained by storing absolute samples every $O(\log n)$ entries and differential encoding for the others, and hence several complex four-Russians-based structures are needed to access between samples in constant time.

Binary Search on Absolute Samples. Our binary searches inside $R(p_i)$, instead, could proceed over the absolute samples first. When the correct interval between samples has been found, we decode the $O(\log n)$ entries sequentially until finding the appropriate entry. The complexity is still $O(\log n)$ per binary search (that is, $O(\log n)$ accesses for the initial binary search plus $O(\log n)$ for the final sequential search), and no extra structure is needed to access Ψ (or Ψ^ℓ). The search is illustrated in Fig. 4.

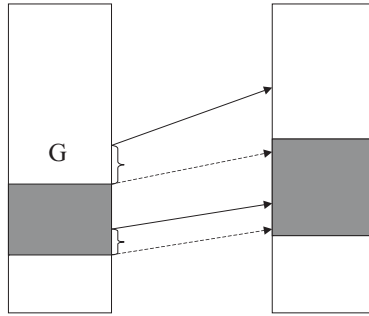


Fig. 4. Binary search followed by sequential search. The top-most sampled value closest to the previous interval is found using binary search (indicated by the top-most solid arrow). Then the next Ψ values are decoded until the first value inside the interval (if exists) is encountered (indicated by the top-most dashed arrow). The same is repeated to find the bottom-most sampled value and then the bottom-most encoded value

We first consider how Ψ can be encoded. We store $\frac{\varepsilon n}{2 \log n} = O(n/\log n)$ absolute samples of Ψ . For each such sample, we need to store value Ψ in $\log n$ bits, as well as a pointer to its corresponding position in the differentially encoded Ψ sequence, in other $\log n$ bits. Overall, the absolute samples require εn bits, for any $\varepsilon > 0$, and permit doing each binary search in $\log n + \frac{2}{\varepsilon} \log n = O(\log n)$ steps. On the other hand, array C needs $o(n)$ bits by choosing any $t > 1$ for its $O(n/\log^t n)$ entries.

The most important issue is how to efficiently encode the differences between consecutive Ψ cells. The $n(H_0 + O(\log \log \sigma))$ space complexity of the original CSA is due to the need of constant time access inside absolute samples, which forces the use of a zero-order compressor. In our case, we could use any compression mechanism between absolute samples, because they will be decoded sequentially.

Compressing Ψ Using Elias Encoding. We now give a simple encoding that slightly improves the space complexity of the original CSA.

The differences $\Psi(i) - \Psi(i - 1)$ can be encoded efficiently using Elias delta coding [26]. Let $b(p)$ be the binary string representation of a number p . We use $1^{b(r)}|0b(r)b(p)$ to encode p , where $r = |b(p)|$. The length of the encoding is $\log(2 \log p + 1) + 1 + \log p = \log p(1 + o(1))$. The overall length of the codes for all differences can be bounded using the following observation: The Ψ values are increasing inside a suffix array region corresponding to a character c . In other words,

$$\sum_{i,i-1 \in R(c)} |\Psi(i) - \Psi(i - 1)| = \sum_{i,i-1 \in R(c)} \Psi(i) - \Psi(i - 1) \leq n. \tag{1}$$

To encode the differences for character c , we thus need $\sum_{i,i-1 \in R(c)} (1 + o(1)) \log(\Psi(i) - \Psi(i - 1))$ bits. This becomes $|R(c)|(1 + o(1)) \log(n/|R(c)|)$ in the worst case, where $|R(c)| = r - \ell + 1$ is the length of the range $R(c) = [\ell, r]$.

Summing over all characters gives

$$\sum_{c \in \Sigma} |R(c)|(1 + o(1)) \log(n/|R(c)|) = nH_0(1 + o(1)). \tag{2}$$

Hence the overall structure occupies $n(H_0 + \varepsilon)(1 + o(1))$ bits.

Also, the “small additional structures” mentioned in Section 3, used to find in constant time the character c such that $C[c] < i \leq C[c + 1]$, are not anymore necessary. These also made use of four-Russians techniques.

Let us consider how to decode a number p coded with Elias. We can read $1^{b(r)}|0$ bitwise in $O(|b(r)|) = O(\log r) = O(\log |b(p)|) = O(\log \log p) = O(\log \log n)$ time. Then we get $b(r)$ and $b(p)$ in constant time. The complexity can be lowered to $O(\log \log \log n)$ if we code r as $1^{b(r')}0b(r')b(r)$, where $r' = |b(r)|$. Indeed, we can apply this until the number to code is zero, for a complexity of $O(\log^* n)$. Alternatively, we can decode Elias in constant time by only small abuse of four-Russians technique: Precompute the first zero of any sequence of length $\log \log n$ and search the table with the first bits of $1^{b(r)}|0b(r)b(p)$. This table needs only $O(\log n \log \log n)$ space.

Due the lack of space we omit the analysis for encoding Ψ^ℓ . In the full version we show that Ψ^ℓ can be encoded to $n \sum_{0 \leq i < \ell} H_i$ bits. We also give an alternative encoding for Ψ combining run-length encoding and Elias encoding to achieve a $2n(H_k(H + \log \log n) + \varepsilon)(1 + o(1))$ bits representation of the structure. (Meanwhile, these analyses appear in a technical report [15–Chapter 5].)

7 A Secondary Memory Implementation

We show now how the regular access pattern can be exploited to obtain an efficient implementation in secondary memory, where the disk blocks can accommodate $B \log n$ bits.

Let us consider again the absolute samples. We pack all the $O(n/\log n)$ absolute samples together, using $O(n/(B \log n))$ blocks. However, the samples are stored in a level-by-level order of the induced binary search tree: For each character c , we store the root of the binary search hierarchy corresponding to the area $C[c] + 1 \dots C[c + 1]$, that is, $\Psi(\lfloor ((C[c] + 1) + C[c + 1])/2 \rfloor)$. Then we store the roots of the left and right children, and so on. When the disk block is filled, the subtrees are recursively packed into disk pages. Fig. 5 illustrates.

Using this arrangement, any binary search inside the area of a character c can make the first $\log B$ accesses by reading only the first disk block. Each new disk block read permits making other $\log B$ accesses. Overall, we find the interval between two consecutive samples in $O(\log(n)/\log(B)) = O(\log_B n)$ disk accesses.

The compressed entries of Ψ are stored in contiguous disk pages. Once we determine the interval between consecutive samples, we sequentially read all the necessary disk pages. This requires reading $O(\log(n)/B)$ additional disk pages, which contributes a lower order term to the cost.

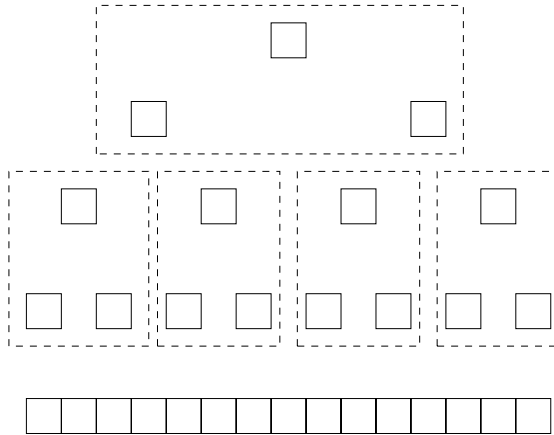


Fig. 5. Packing of array cells to optimize binary search in secondary memory. The dashed boxes indicate cells mapped to a disk block. Any binary search on the array at the bottom is carried out with 2 disk accesses

Overall, we can maintain the CSA on disk and search it in $O(m \log_B n)$ disk accesses. The original structure requires $O(m \log n)$ disk accesses. If we can hold $O(n)$ bits in main memory, then we can cache all the absolute samples and pay only $O(m \log(n)/B)$ disk accesses.

This scheme can be extended to use a table C of ℓ -grams rather than of individual characters. Each individual binary search takes still $O(\log_B n)$ time, but we perform only $\lceil m/\ell \rceil$ of them.

One obstacle to a secondary memory CSA implementation might be in building such a large CSA. This issue has been addressed satisfactorily [21].

8 A Distributed Implementation

Distributed implementations of suffix arrays face the problem that not only the suffix array, but also the text, are distributed. Hence, even if we distribute suffix array \mathcal{A} according to lexicographical intervals, the processor performing the local binary search will require access to remote text positions [17]. Although some heuristics have been proposed, $\log n$ remote requests for m characters each are necessary in the worst case.

The original CSA does not help solve this. If array Ψ is distributed, we will need to request cells of Ψ to other processors for each character of each binary search step, for a total of $m \log n$ remote requests. Actually this is worse than $\log n$ requests for m characters each.

The backward search mechanism permits us to do better. Say that one processor is responsible for the interval corresponding to each character. Then, we can process pattern characters p_m, p_{m-1}, \dots, p_1 as follows: The processor responsible for p_m sends $R(p_m)$ to the processor responsible for p_{m-1} . That processor

binary searches in its local memory for the cells that point inside $R(p_m)$, without any communication need. Upon completing the search, it sends $R(p_{m-1}p_m)$ to the processor responsible for p_{m-2} and so on. After m communication steps exchanging $O(1)$ data, we have the answer.

In the BSP model [24], we need m supersteps of $O(\log n)$ CPU work and $O(1)$ communication each. In comparison, the CSA needs $O(m \log n)$ supersteps of $O(1)$ CPU and communication each, and the basic suffix array needs $O(\log n)$ supersteps of $O(m)$ CPU and communication each.

More or less processors can be accommodated by coarsening or refining the lexicographical intervals. Although the real number of processors, r , is irrelevant to search for one pattern (note that the elapsed time is still $O(m \log n)$), it becomes important when performing a sequence of searches.

If N search patterns, each of length m , are entered into the system, and we assume that the pattern characters distribute uniformly over Σ , then a pipelining effect occurs. That is, the processor responsible for p_m becomes idle after the first superstep and it can work on subsequent patterns. On average the N answers are obtained after Nm/r supersteps and $O(Nm \log(n)/r)$ total CPU work, with $O(Nm/r)$ communication work.

Hence, we have obtained $O(m \log(n)/r)$ amortized time per pattern with r processors, which is the optimal speedup over the sequential algorithm. The communication effort is $O(m/r)$, of lower order than the computation effort. We can apply again the technique of Section 5 to reduce the CPU time to $O(\lceil m/\ell \rceil \log n/r)$.

9 Conclusions

We have proposed a new implementation of the backward search algorithm for the Compressed Suffix Array (CSA). The new method takes advantage of the regular access pattern to the structure, which allows several improvements over the CSA: (i) tradeoff between search time and space requirement, (ii) simpler and more compact structure implementation, (iii) efficient secondary memory implementation, and (iv) efficient distributed implementation. In particular, ours is the only succinct full-text index structure not relying on four-Russians techniques.

References

1. V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economic construction of the transitive closure of a directed graph. *Dokl. Acad. Nauk. SSSR* 194, 487–488, 1970 (in Russian). English translation in *Soviet Math. Dokl.* 11, 1209–1210, 1975.
2. D. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1996.
3. P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. FOCS'00*, pp. 390–398, 2000.

4. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. An Alphabet-Friendly FM-index. To appear in *11th Symposium on String Processing and Information Retrieval (SPIRE 2004)*, Padova, Italy, October 5-8, 2004.
5. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. SODA'03*, pp. 841–850, 2003.
6. R. Grossi, A. Gupta, and J. Vitter. When indexing equals compression: Experiments with compressing suffix arrays and applications. In *Proc. SODA'04*, pp. 636-645, 2004.
7. R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. STOC'00*, pp. 397–406, 2000.
8. G. Jacobson. *Succinct Static Data Structures*. PhD thesis, CMU-CS-89-112, Carnegie Mellon University, 1989.
9. J. Kärkkäinen. *Repetition-Based Text Indexes*, PhD Thesis, Report A-1999-4, Department of Computer Science, University of Helsinki, Finland, 1999.
10. U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22, pp. 935–948, 1993.
11. G. Manzini. An Analysis of the Burrows-Wheeler Transform. *J. of the ACM* 48(3):407–430, 2001.
12. E. M. McCreight. A space economical suffix tree construction algorithm. *J. of the ACM*, 23, pp. 262–272, 1976.
13. I. Munro. Tables. In *Proc. FSTTCS'96*, pp. 37–42, 1996.
14. V. Mäkinen. Compact Suffix Array — A space-efficient full-text index. *Fundamenta Informaticae* 56(1-2), pp. 191–210, 2003.
15. V. Mäkinen and G. Navarro. New search algorithms and space/time tradeoffs for succinct suffix arrays. Technical report, C-2004-20, Dept. CS, Univ. Helsinki, April 2004. [http://www.cs.helsinki.fi/u/vmakinen/papers/ssa_tech_2004.ps.gz]
16. V. Mäkinen and G. Navarro. Compressed compact suffix arrays. In *Proc. CPM'04*, LNCS 3109, pp. 420–433, 2004.
17. M. Marín and G. Navarro. Distributed query processing using suffix arrays. In *Proc. SPIRE'03*, pages 311–325, LNCS 2857, 2003.
18. G. Navarro. Indexing text using the Ziv-Lempel trie. *J. of Discrete Algorithms* 2(1):87–114, 2004.
19. K. Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proc. ISAAC'00*, LNCS 1969, pp. 410–421, 2000.
20. K. Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *Proc. SODA 2002*, ACM-SIAM, pp. 225–232, 2002.
21. K. Sadakane. Constructing compressed suffix arrays with large alphabets. In *Proc. ISAAC'03*, LNCS 2906, pp. 240–249, 2003.
22. S. Srinivasa Rao. Time-space trade-offs for compressed suffix arrays. *Inf. Proc. Lett.*, 82 (6), pp. 307-311, 2002.
23. E. Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14, pp. 249–260, 1995.
24. L. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33:103–111, Aug. 1990.
25. P. Weiner. Linear pattern matching algorithms. In *Proc. IEEE 14th Ann. Symp. on Switching and Automata Theory*, pp. 1–11, 1973.
26. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, New York, second edition, 1999.

Inner Rectangular Drawings of Plane Graphs (Extended Abstract)

Kazuyuki Miura, Hiroki Haga, and Takao Nishizeki

Graduate School of Information Sciences,
Tohoku University, Sendai 980-8579, Japan
{miura, haga}@nishizeki.ecei.tohoku.ac.jp
nishi@ecei.tohoku.ac.jp

Abstract. A drawing of a plane graph is called an inner rectangular drawing if every edge is drawn as a horizontal or vertical line segment so that every inner face is a rectangle. In this paper we show that a plane graph G has an inner rectangular drawing D if and only if a new bipartite graph constructed from G has a perfect matching. We also show that D can be found in time $O(n^{1.5}/\log n)$ if G has n vertices and a sketch of the outer face is prescribed, that is, all the convex outer vertices and concave ones are prescribed.

1 Introduction

A drawing of a plane graph is called a *rectangular drawing* if every edge is drawn as a horizontal or vertical line segment so that every face boundary is a rectangle. Figures 1(a) and 2(d) illustrate rectangular drawings. A rectangular drawing often appears in VLSI floor-planning and architectural layout [DETT99, FW74, GT97, Len90, SY99]. Each inner face of a plane graph represents a module of a VLSI circuit or a room of an architectural layout. Suppose that a plane graph G_a representing the requirement of adjacency among modules is given as illustrated in Fig. 2(a). Each vertex of G_a corresponds to a module of a VLSI circuit, and each edge of G_a means that the two modules corresponding to the ends are required to be adjacent in the VLSI floor-planning, that is, the two rectangular modules must share a common boundary. A conventional method obtains a floor plan meeting the adjacency requirement represented by G_a , as follows:

- (1) add dummy edges to G_a so that every inner face of the resulting graph G'_a is a triangle, as illustrated in Fig. 2(b) where dummy edges are drawn by dotted lines;
- (2) construct a new plane graph G''_a from a dual-like graph of G'_a by adding four vertices of degree two corresponding to the four corners of the rectangular chip, as illustrated in Fig. 2(c) where G'_a is drawn by dotted lines, G''_a by solid lines, and the four added vertices by white circles;
- (3) find a rectangular drawing of G''_a as a floor plan meeting the requirement of G_a , as illustrated in Fig. 2(d).

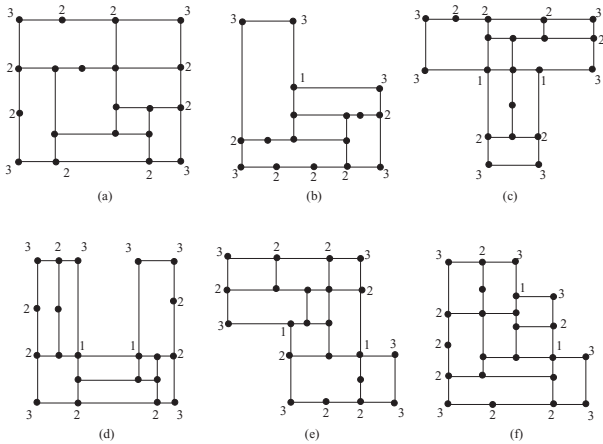


Fig. 1. Rectangular drawing (a), and inner rectangular drawings of (b) L-shape, (c) T-shape, (d) U-shape, (e) Z-shape, and (f) staircase-shape

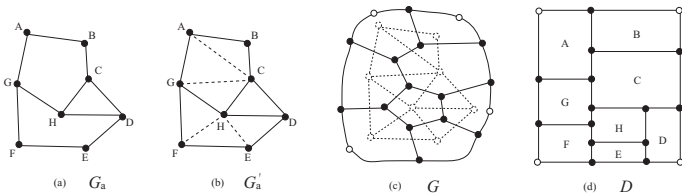


Fig. 2. (a) Adjacency requirement graph G_a , (b) inner triangulated graph G'_a augmented from G_a , (c) dual-like graph G of G'_a , and (d) rectangular drawing D of G with $\Delta = 3$

In the plane graph appearing in the conventional method above, all vertices have degree three except for the four vertices of degree two corresponding to the four corners, because every inner face of G'_a is a triangle. Hence the maximum degree Δ of G is three. However, some plane graphs with $\Delta = 4$ may have rectangular drawing, as illustrated in Fig. 3(d). Of course, Δ must be four or less if G has a rectangular drawing. A necessary and sufficient condition for a plane graph with $\Delta \leq 3$ to have a rectangular drawing is known, and linear or $O(n^{2.5}/\log n)$ algorithms to find a rectangular drawing of G is obtained [BS88, He93, KH97, KK84, LL90, RNN02, RNN98, Tho84]. However, it has been an open problem to obtain a necessary and sufficient condition and an efficient algorithm for plane graphs with $\Delta \leq 4$ [MMN02, RNN98].

Let G''_a be a plane graph obtained from an adjacency requirement graph G_a by adding dummy edges to G_a so that every inner face is either a triangle or a quadrilateral, as illustrated in Fig. 3(b). Let G'' be a plane graph obtained from a dual-like graph of G''_a by adding four vertices of degree two corresponding to the corners. Of course, the maximum degree Δ of G'' may be four since G''_a may

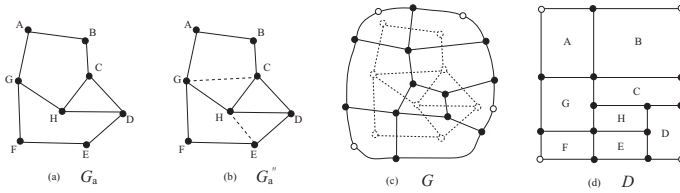


Fig. 3. (a) Adjacency requirement graph G_a , (b) inner triangulated or quadrangulated graph G_a'' augmented from G_a , (c) dual-like graph G of G_a'' , and (d) rectangular drawing D of G with $\Delta = 4$

have a quadrangular inner face. If one can find a rectangular drawing of G with $\Delta \leq 4$ as illustrated in Fig. 3(d), then one can use D as a floor plan meeting the requirement of G_a .

The outer face boundary must be rectangular in a rectangular drawing, as illustrated in Fig. 1(a). However, the outer boundary of a VLSI chip or an architectural floor plan is not always rectangular, but is often a rectilinear polygon of L-shape, T-shape, U-shape or staircase-shape, as illustrated in Figs. 1(b)–(f) [FW74, Len90, MMN02, SS93, YS93]. We call such a drawing of a plane graph an *inner rectangular drawing* if every inner face of G is a rectangle although the outer face boundary is not always a rectangle.

In the paper we show that a plane graph G has an inner rectangular drawing if and only if a new bipartite graph constructed from G has a perfect matching. We also show that G can be found in time $O(n^{1.5}/\log n)$ if a “sketch” of the outer face is prescribed, that is, all the convex outer vertices and concave ones are prescribed, where n is the number of vertices of G . We do not assume $\Delta \leq 3$, and an inner rectangular drawing is a rectangular drawing if the outer face is sketched as a rectangle. Thus we solve the open problem above.

The remainder of the paper is organized as follows. In Section 2 we define some terms, describe some fundamental facts, and present our main results, Theorems 1–3. In Section 3 we prove Theorem 1 for the case where a sketch of the outer face is prescribed. In Section 4 we prove Theorem 2 for the case where the numbers of “convex” and “concave” outer vertices are prescribed. In Section 5 we prove Theorem 3 for a general case.

2 Preliminaries and Main Results

We assume in the paper that G is a plane undirected simple graph. We denote by $d(v)$ the degree of a vertex v in G . We denote by ∂G the outer face of G . The boundary of ∂G is called the *outer boundary*, and is denoted also by ∂G . A vertex on ∂G is called an *outer vertex*, and a vertex not on ∂G is called an *inner vertex*. We may assume without loss of generality that G is 2-connected and $\Delta \leq 4$, and hence every vertex of G has degree 2, 3 or 4.

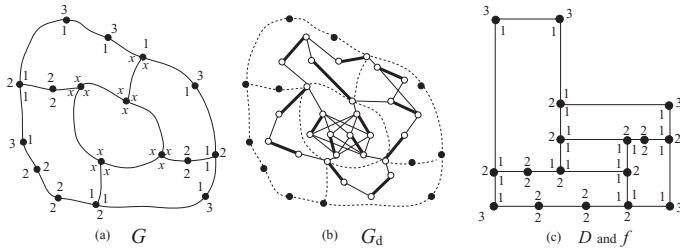


Fig. 4. (a) Plane graph G , (b) decision graph G_d , and (c) inner rectangular drawing D and regular labeling f of G

An angle formed by two edges e and e' incident to a vertex v in G is called an *angle* of v if e and e' appear consecutively around v . An angle of a vertex in G is called an *outer angle* if v is on the boundary of a face in G . An angle formed by two consecutive edges on a boundary of a face in G is called an *inner angle*. An angle of the outer face is called an *outer angle* of G , while an angle of an inner face is called an *inner angle* of G .

In any inner rectangular drawing, every inner angle is $\pi/2$ or π , and every outer angle is $\pi/2$, π or $3\pi/2$. Consider a labeling f which assigns a label 1, 2, or 3 to every angle of G , as illustrated in Fig. 4(c). Labels 1, 2 and 3 correspond to angles $\pi/2$, π and $3\pi/2$, respectively. We denote by n_{cv} the number of outer angles having label 3, and by n_{cc} the number of outer angles having label 1. Thus n_{cv} is the number of “convex” outer vertices, and n_{cc} is the number of “concave” outer vertices. For example $n_{cv} = 5$ and $n_{cc} = 1$ for the labeling f in Fig. 4(c).

We call a *regular labeling* of G if f satisfies the following three conditions (a)–(c):

- (a) For each vertex v of G , the labels of all the angles of v total to 4;
- (b) The label of any inner angle is 1 or 2, and every inner face has exactly four angles of label 1; and
- (c) $n_{cv} - n_{cc} = 4$.

Figure 4(c) illustrates a regular labeling f of the plane graph in Fig. 4(a) and an inner rectangular drawing D corresponding to f .

Conditions (a) and (b) imply the following (i)–(iii):

- (i) If a vertex v has degree 2, that is, $d(v) = 2$, then the two labels of v are either 2 and 2 or 1 and 3. In particular, if v is an inner vertex, then the two labels are 2 and 2.
- (ii) If $d(v) = 3$, then exactly one of the three angles of v has label 2 and the other two have label 1.
- (iii) If $d(v) = 4$, then all the four angles of v have label 1.

If G has an inner rectangular drawing, then clearly G has a regular labeling. Conversely, if G has a regular labeling, then G has an inner rectangular drawing, as can be proved by means of elementary geometric considerations. We thus have the following fact.

Fact 1. *Let G be a plane graph with n vertices and m edges. Then G has a regular labeling if and only if $m \leq 3n - 6$.*

A drawing of a plane graph is called an *orthogonal drawing* if each edge is drawn as an alternating sequence of horizontal and vertical line segments. A point at which an edge changes its direction is called a *bend*. An inner rectangular drawing is an orthogonal drawing with no bend such that every inner face is a rectangle. Tamassia gives an algorithm to find an orthogonal drawing of a given plane graph with the minimum number of bends in time $O(n^2 \log n)$ by solving the “minimum cost flow problem” of a new graph constructed from G [Tam87]. Garg and Tamassia refine the algorithm so that it takes time $O(n^{1.75} \sqrt{\log n})$ [GT97]. Tamassia presents an “orthogonal representation” of a plane graph for characterizing an orthogonal drawing [DETT99, Tam87]. Our regular labeling can be regarded as a special case of his orthogonal representation. He gives also a linear algorithm to find an orthogonal “grid” drawing from an orthogonal representation, in which every vertex has an integer coordinate. Similarly, one can find an inner rectangular “grid” drawing from a regular labeling in linear time.

A *regular labeling* of the outer face F_o of a plane graph G is to assign a label 1, 2 or 3 to the outer angle α of each outer vertex v of G , as illustrated in Fig. 1. If $d(v) = 2$, then the label of α must be either 2 or 3. If $d(v) = 3$, then the label must be either 1 or 2. If $d(v) = 4$, then the label must be 1. Furthermore, $\ell_{cv} - \ell_{cc} = 4$. For example, $\ell_{cv} = 5$ and $\ell_{cc} = 1$ for the sketches in Figs. 1(b) and 4(a), and hence the sketches imply that the outer face boundary F_o must have an L-shape.

Suppose first that a sketch of F_o is prescribed. Then one can immediately determine some of the inner angles by Conditions (a) and (b) of a regular labeling, as illustrated in Fig. 4(a). The remaining undetermined inner angles are labeled with ℓ , which means either 1 or 2. We construct from G a new graph G_d , called a *drawn graph* of G , as illustrated in Fig. 4(b) where G_d is drawn by solid lines and G by dotted lines. We then have the following theorem.

Theorem 1. *Let G be a plane graph with n vertices and m edges. Let G_d be a drawn graph of G with n_d vertices and m_d edges. Then G has a regular labeling if and only if $m_d \leq 3n_d - 6$ and $m \leq 3n - 6 + O(n^{1.5}/\log n)$.*

The proof of Theorem 1 and together with the construction of G_d will be given in Section 3.

Suppose next that a sketch of F_o is not prescribed but the numbers ℓ_{cv} and ℓ_{cc} of convex and concave outer vertices are prescribed. Of course, $\ell_{cv} - \ell_{cc} = 4$. For example, $\ell_{cv} = 4$ and $\ell_{cc} = 0$ mean that F_o is rectangular as illustrated in Fig. 1(a), while $\ell_{cv} = 6$ and $\ell_{cc} = 2$ mean that F_o has a T-shape, U-shape, Z-shape or staircase-shape as illustrated in Figs. 1(c)–(f). Let G_d^* be a graph constructed from G , as illustrated in Fig. 6(b). Then we have the following theorem.

Theorem 2.

Let G be a plane graph with n vertices and m edges. Let n_{cv} and n_{cc} be the number of vertices of degree d with d outer angles labeled with 3 and 1, respectively. Let G_d^* be a graph constructed from G . Then we have the following theorem.

$$O(\sqrt{n}(n + n_{cv}n_o)/\log n)$$

The proof of Theorem 2 together with the construction of G_d^* will be given in Section 4.

Consider finally a general case where neither a sketch of F_o nor a pair (n_{cv}, n_{cc}) is prescribed. Let G_d^* be a graph constructed from G . Then we have the following theorem.

Theorem 3.

Let G be a plane graph with n vertices and m edges. Let n_{o2} and n_{o4} be the number of vertices of degree 2 and 4, respectively. Let G_d^* be a graph constructed from G . Then we have the following theorem.

$$O(\sqrt{n}(n + (n_{o2} - n_{o4})n_o)/\log n)$$

The proof of Theorem 3 together with the construction of G_d^* will be given in Section 5.

We assume that some trivial conditions for the existence of an inner rectangular drawing hold in Theorems 1, 2 and 3, as we will explain in Sections 3, 4 and 5.

3 Inner Rectangular Drawing with Sketched Outer Face

In this section we prove Theorem 1.

Suppose that a sketch of the outer face of a plane graph G is prescribed, that is, all the outer angles of G are labeled with 1, 2 or 3, as illustrated in Figs. 1 and 4(a). Of course, the number n_{cv} of outer angles labeled with 3 and the number n_{cc} of outer angles labeled with 1 must satisfy $n_{cv} - n_{cc} = 4$. The outer angle of an outer vertex v must be labeled with either 2 or 3 if $d(v) = 2$, with either 1 or 2 if $d(v) = 3$, and with 1 if $d(v) = 4$. Then some of the inner angles of G can be immediately determined, as illustrated in Fig. 4(a). If v is an outer vertex of degree 2 and the outer angle of v is labeled with 2 and 3, then the inner angle of v must be labeled with 2 and 1, respectively. The two angles of any inner vertex of degree 2 must be labeled with 2. If v is an outer vertex of degree 3 and the outer angle of v is labeled with 2, then both of the inner angles of v must be labeled with 1. On the other hand, if v is an outer vertex of degree 3 and the outer angle of v is labeled with 1, then we label both of the inner angles of v with \cdot , because one cannot determine their labels at this moment although one of them must be labeled with 1 and the other with 2. We also label all the three angles of an inner vertex of degree 3 with \cdot , because one cannot determine their labels although exactly one of them must be labeled with 2 and the others with 1. We label all the four angles of each vertex of degree 4 with 1. Label \cdot means that \cdot is either 1 or 2, and exactly one of the labels \cdot 's attached to the same vertex must be 2 and the others must be 1. (See Figs. 4(a) and (c).)

We now present how to construct a decision graph G_d of G . Let all vertices of G attached a label l be vertices of G_d . Thus all the inner vertices of degree 3 and all the outer vertices of degree 3 whose outer angles are labeled with 1 are vertices of G_d , and none of the other vertices of G is a vertex of G_d . We then add to G_d a complete bipartite graph inside each inner face f of G . Let n_x be the number of angles of f labeled with 1 . Let n_1 be the number of angles of G which have been labeled with 1. One may assume that $n_1 \leq 4$; otherwise, G has no inner rectangular drawing. Exactly $4 - n_1$ of the n_x angles of f labeled with 1 must be labeled with 1 by a regular labeling. We add a complete bipartite graph $K_{(4-n_1)n_x}$ in f , and join each of the n_x vertices in the second partite set with one of the n_x vertices on f whose angles are labeled with 1 . Repeat the operation above for each inner face f of G . The resulting graph is a decision graph G_d of G . The decision graph G_d of the plane graph G in Fig. 4(a) is drawn by solid lines in Fig. 4(b), where G is drawn by dotted lines. The idea of adding a complete bipartite graph originates from Tutte's transformation for finding an " n -factor" of a graph [Tut54].

A matching of G_d is a set of pairwise non-adjacent edges in G_d . A maximum matching of G_d is a matching of the maximum cardinality. A matching of G_d is called a perfect matching if an edge in G_d is incident to each vertex of G_d . A perfect matching is drawn by thick solid lines in Fig. 4(b).

Each edge of G_d incident to a vertex v attached a label l corresponds to an angle α of G labeled with l . A fact that v is contained in a perfect matching of G_d means that the label l of α is 2. Conversely, a fact that v is not contained in M means that the label l of α is 1.

We are now ready to prove Theorem 1.

[Proof of Theorem 1]

Suppose that G has an inner rectangular drawing with a sketched outer face. Then by Fact 1 G has a regular labeling l which is an extension of the sketch, that is, a labeling of the outer angles. We include in a set M all the edges of G_d corresponding to angles of label $l = 2$, while we do not include in M the edges of G_d corresponding to angles of label $l = 1$. For each vertex v of G_d attached a label l , the labeling l assigns 2 to exactly one of the angles of f labeled with l . Therefore exactly one of the edges of G_d incident to v is contained in M . The labeling l labels exactly four of the angles of each inner face f with 1. Therefore exactly $4 - n_1$ of the n_x angles of f labeled with 1 must be labeled with 1 by l , and hence all the edges of G_d corresponding to these angles are not contained in M . Including in M a number $4 - n_1$ of edges in each complete bipartite graph, we can extend M to a perfect matching of G_d . Thus G_d has a perfect matching.

Conversely, if G_d has a perfect matching, then G has a regular labeling which is an extension of a sketch of the outer face, and hence by Fact 1 G has an inner rectangular drawing with a sketched outer face.

Clearly, G_d is a bipartite graph, and $4 - n_1 \leq 4$. Obviously, n_x is no more than the number of edges on face f . Let m be the number of edges in G , then we have $2m \leq 4n$ since $\Delta \leq 4$. Therefore the sum $2m$ of the numbers of edges on

all faces is at most $4n$. One can thus know that both the number ν_d of vertices in \mathcal{F}_d and the number e_d of edges in \mathcal{F}_d are $O(n)$. Since \mathcal{F}_d is a bipartite graph, a maximum matching of \mathcal{F}_d can be found either in time $O(\sqrt{e_d} \nu_d) = O(n^{1.5})$ by an ordinary bipartite matching algorithm [HK73, MV80, PS82] or in time $O(\sqrt{n_d} m_d / \log n_d) = O(n^{1.5} / \log n)$ by a recent pseudoflow-based bipartite matching algorithm using boolean word operations on $\log n$ -bit words [FM91, Hoc04, HC04].

This completes a proof of Theorem 1. □

Lai and Leinwand show that a plane graph G with $\Delta \leq 3$ has a rectangular drawing if and only if a new bipartite graph constructed from G and its dual has a perfect matching [LL90]. Their bipartite graph has an $O(n)$ number of vertices, but has an $O(n^2)$ number of edges. Therefore their method takes time $O(n^{2.5} / \log n)$ to find a rectangular drawing of G .

We also remark that one can enumerate all inner rectangular drawings of G by enumerating all perfect matchings of \mathcal{F}_d .

Clearly the bipartite matching problem can be reduced to the maximum flow problem [AMO93, PS82]. Conversely, modifying Tamassia's formulation of the bend-minimum orthogonal drawing problem [Tam87], one can directly reduce the inner rectangular drawing problem to a flow problem on a new planar bipartite network N with multiple sources and sinks. A network N for the plane graph G in Fig. 4(a) is illustrated in Fig. 5; N is drawn by solid lines, a flow value is attached to each arc, and an inner rectangular drawing of G corresponding to the flow is drawn by dotted lines. Every arc of N has a capacity 1. A node of N corresponding to a vertex of G is a source, the supply of which is written in a circle in Fig. 5. A node of N corresponding to an inner face of G is a sink, the demand of which is written in a square in Fig. 5. An inner angle of $\pi/2$ is represented by an arc of flow 1, while an inner angle of π is represented by an arc of flow 0. One can observe that G has an inner rectangular drawing with the sketched outer face if and only if N has a feasible (single commodity) flow satisfying all the demands. A feasible flow in such a planar network or a bipartite network can be found either in time $O(n^{1.5})$ by a planar flow algorithm [MN95] or a bipartite flow algorithm [AMO93, ET75] or in time $O(n^{1.5} / \log n)$ by the pseudoflow-based bipartite flow algorithm [Hoc04, HC04].

Thus both our matching approach and the flow approach solve the inner rectangular drawing problem in the same time complexity. However, the bipartite matching algorithm in [HK73] can be quite easily implemented in comparison with the flow algorithms in [Hoc04, HC04, MN95].

4 Inner Rectangular Drawing with Prescribed Numbers n_{cv} and n_{cc}

In this section we prove Theorem 2.

Suppose that an outer face of a plane graph G is not sketched but a pair (n_{cv}, n_{cc}) is prescribed, where n_{cv} is the number of convex outer vertices and n_{cc} is the number of concave outer vertices. If (n_{cv}, n_{cc}) is prescribed as $n_{cv} = 5$

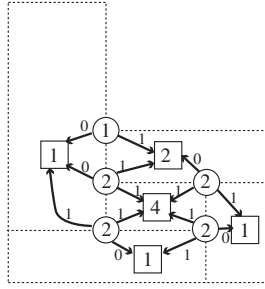


Fig. 5. Network N

and $\alpha_{cc} = 1$ like in Fig. 6(a), then the outer boundary must have an L-shape as illustrated in Fig. 6(c), but it has not been prescribed which outer vertices are convex and which outer vertices are concave. We label all the four angles of each vertex of degree 4 in G with 1, and label both of the angles of each inner vertex of degree 2 with 2. The labels of all the other angles of G are not determined at this moment, and we label them with α or β as follows:

- (1) label all the three angles of any vertex v of degree 3 with α ; and
- (2) label the inner angle of any outer vertex v of degree 2 with α , and label the outer angle of v with β .

The labeling of the same plane graph as in Fig. 4(a) is depicted in Fig. 6(a). Label α means that α is either 1 or 2, similarly as in Section 3. On the other hand, label β means that β is either 2 or 3. Each outer vertex of degree 2 is attached two labels α and β , and if $\alpha = 1$ then $\beta = 3$ and if $\alpha = 2$ then $\beta = 2$.

We now present how to construct a decision graph G_d^* of G . The construction is similar to that of G_d . Let all the vertices of G attached label α or β be vertices of G_d^* as illustrated in Fig. 6(b). Thus all the outer vertices of degree 2 in G and all the vertices of degree 3 in G are vertices of G_d^* . All the other vertices of G are not vertices of G_d^* : all the vertices of degree 4 and all the inner vertices of degree 2 are not vertices of G_d^* .

For each inner face f of G , we add a complete bipartite graph $K_{(4-n_1)n_x}$ inside f , where n_x is the number of angles of f labeled with α and n_1 is the number of angles of f labeled with 1. Of course, one may assume that $n_1 \leq 4$.

We then add two complete bipartite graphs inside the outer face f_o , as follows. Let α_{ox} be the number of outer angles of f_o labeled with α , and let α_{oy} be the number of outer angles labeled with β . For the example in Fig. 6(a) $\alpha_{ox} = 4$ and $\alpha_{oy} = 7$. Let α_{o4} be the number of outer vertices v of degree 4. For the example in Fig. 6(a) $\alpha_{o4} = 0$. The outer angle of v must be labeled with 1, and v must be a concave outer vertex. One may assume without loss of generality that $n_{cc} \geq n_{o4}$; otherwise, G has no inner rectangular drawing with n_{cc} concave outer vertices. Exactly $\alpha_{cc} - \alpha_{o4}$ of the n_{ox} outer angles of label α must be labeled with 1. We add a complete bipartite graph $K_{(n_{cc}-n_{o4})n_{ox}}$ in F_o , and joint each of the n_{ox} vertices in the second partite set with one of the α_{ox} outer vertices of

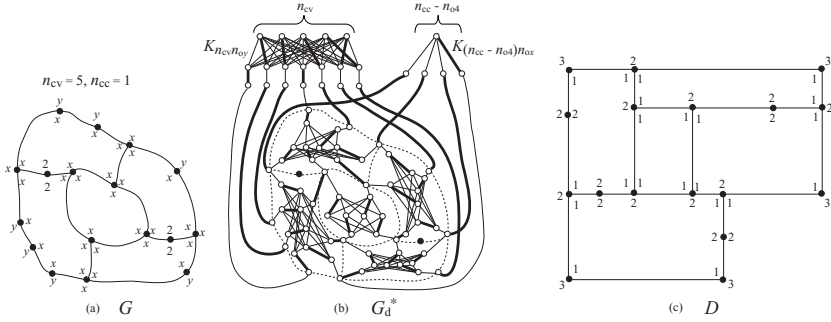


Fig. 6. (a) Plane graph G and pair (n_{cv}, n_{cc}) , (b) decision graph G_d^* , and (c) inner rectangular drawing D and regular labeling f of G

degree 3. Exactly n_{cv} of the n_{oy} outer angles of label \cdot must be labeled with 3. We add a complete bipartite graph $K_{n_{cv}n_{oy}}$ inside F_o , and connect each of the n_{oy} vertices in the second partite set with one of the n_{oy} outer vertices of degree 2 via a path of length 2.

This completes a construction of G_d^* . In Fig. 6(b) G_d^* is drawn by solid lines, and F_o by dotted lines. A perfect matching of G_d^* is drawn by thick solid lines in Fig. 6(b).

Let α be an angle of a vertex v of G labeled with \cdot or \cdot , and let e be the edge of G_d^* which is incident to v and corresponds to α . A fact that e is contained in a perfect matching of G_d^* means that the label of α is 2 if it is \cdot and the label is 3 if it is \cdot . Conversely, a fact that e is not contained in M means that the label of α is 1 if it is \cdot and the label is 2 if it is \cdot . Similarly as for Theorem 1, one can easily prove that G has an inner rectangular drawing with \cdot_{cv} convex outer vertices and \cdot_{cc} concave ones if and only if G_d^* has a perfect matching. Clearly G_d^* is a bipartite graph. Since $n_o \geq \cdot_{cv} > \cdot_{cc}$, G_d^* has an $O(n)$ number of vertices and an $O(n + n_{cv}n_o)$ number of edges, where \cdot_o is the number of outer vertices of G . Thus a maximum matching of G_d^* can be found in time $O(\sqrt{n}(n + n_{cv}n_o)/\log n)$.

This completes a proof of Theorem 2.

5 Inner Rectangular Drawing

In this section we prove Theorem 3.

Suppose that neither a sketch of the outer face F_o nor a pair of integers (\cdot_{cv}, \cdot_{cc}) is prescribed for a plane graph G . Let n_{o2}, n_{o3} and n_{o4} be the numbers of outer vertices having degrees 2, 3 and 4, respectively, then $n_o = n_{o2} + n_{o3} + n_{o4}$. Since $\cdot_{cv} \leq \cdot_{o2}$ and $\cdot_{cv} - \cdot_{cc} = 4$, there are at most a number n_{o2} of pairs which are possible as (\cdot_{cv}, \cdot_{cc}) . Examining all these pairs, one can know whether G has an inner rectangular drawing for some pair. Such a straightforward method would take time $O(n_{o2}\sqrt{n}(n + n_{o2}n_o)/\log n)$. However, we can show as in Theorem 3 that G has an inner rectangular drawing D for some pair if and only if a new

graph G_d^* constructed from G has a perfect matching, and that G can be found in time $O(\sqrt{n}(n + (n_{o_2} - n_{o_4})n_o)/\log n)$ whenever G has \mathcal{P} .

We label each angle of G with 1, 2, x or y in the same way as in Section 4, as illustrated in Fig. 6(a). There are n_{o_4} outer angles labeled with 1, n_{o_3} with x , and n_{o_2} with y . One may assume without loss of generality that $n_{o_2} \geq n_{o_4} + 4$; otherwise, G has no inner rectangular drawing. The construction of a new graph G_d^* is the same as G_d^* except for the outer face F_o . We add a complete bipartite graph $B = K_{(n_{o_2}-n_{o_4}-4)(n_{o_2}+n_{o_3})}$ in F_o , and join each of the $n_{o_2} + n_{o_3}$ vertices in the second partite set of B with one of the $n_{o_2} + n_{o_3}$ outer vertices of degree 2 or 3.

Suppose that G_d^* has a perfect matching M . Let a be the number of edges of G_d^* which correspond to outer angles of outer vertices of degree 2 and are contained in M . Let b be the number of edges of G_d^* which correspond to outer angles of outer vertices of degree 3 and are contained in M . Since M covers all the $n_{o_2} - n_{o_4} - 4$ vertices in the first partite set of B , we have $(n_{o_2} - a) + (n_{o_3} - b) = n_{o_2} - n_{o_4} - 4$, and hence $a + b = n_{o_3} + n_{o_4} + 4$. We assign 2 to the b outer angles of outer vertices which have label x and are ends of edges of M in F_o , and assign 1 to the remaining $(n_{o_3} - b)$ outer angles of label x . We assign 3 to the a outer angles of outer vertices which have label y and are ends of edges of M in F_o , and assign 2 to the remaining $(n_{o_2} - a)$ outer angles of label y . Then we have $n_{cv} = a$ and $n_{cc} = (n_{o_3} - b) + n_{o_4}$, and hence $n_{cv} - n_{cc} = a - (n_{o_3} - b + n_{o_4}) = 4$. One can thus know that G has a regular labeling if G_d^* has a perfect matching.

Conversely G_d^* has a perfect matching if G has an inner rectangular drawing. This completes a proof of Theorem 3.

Acknowledgments. We thank Ayako Miyazawa, Md. Saidur Rahman and Xiao Zhou for fruitful discussions on an early version of the paper.

References

- [AM093] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [BS88] J. Bhasker and S. Sahni, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, *Algorithmica*, 3, pp. 247-278, 1988.
- [DETT99] G. Di Battista, P. Eades, R. Tamassia and G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, NJ, 1999.
- [ET75] S. Even and R. E. Tarjan, *Network flow and testing graph connectivity*, *SIAM J. Computing*, 4, pp. 507-518, 1975.
- [FM91] T. Feder and R. Motowani, *Clique partitions, graph compression and speeding-up algorithms*, Proc. 23rd Ann. ACM Symp. on Theory of Computing, pp. 123-133, 1991.
- [FW74] R. L. Francis and J. A. White, *Facility Layout and Location*, Prentice Hall-CNew Jersey, 1974.
- [GT97] A. Garg and R. Tamassia, *A new minimum cost flow algorithm with applications to graph drawing*, Proc. of Graph Drawing '96, Lect. Notes in Computer Science, Springer, 1190, pp. 201-206, 1997.

- [He93] X. He, *On finding the rectangular duals of planar triangulated graphs*, *SIAM J. Comput.*, 22, 6, pp. 1218-1226, 1993.
- [Hoc04] D. S. Hochbaum, *Faster pseudoflow-based algorithms for the bipartite matching and the closure problems*, Abstract, CORS/SCRO-INFORMS Joint Int. Meeting, Banff, Canada, p. 46, May 16-19, 2004.
- [HC04] D. S. Hochbaum and B. G. Chandran, *Further below the flow decomposition barrier of maximum flow for bipartite matching and maximum closure*, Working paper, 2004.
- [HK73] J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, *SIAM J. Comput.*, 2, pp. 225-231, 1973.
- [KH97] G. Kant and X. He, *Regular edge-labeling of 4-connected plane graphs and its applications in graph drawing problems*, *Theoret. Comput. Sci.*, 172, pp. 175-193, 1997.
- [KK84] K. Kozminski and E. Kinnen, *An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits*, Proc. of 21st DAC, Albuquerque, pp. 655-656, 1984.
- [LL90] Y.-T. Lai and S. M. Leinwand, *A theory of rectangular dual graphs*, *Algorithmica*, 5, pp. 467-483, 1990.
- [Len90] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, Chichester, 1990.
- [Mn95] G. L. Miller and J. S. Naor, *Flows in planar graphs with multiple sources and sinks*, *SIAM J. Computing*, 24(5), pp. 1002-1017, 1995.
- [MV80] S. Micali and V. V. Vazirani, *An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs*, Proc. 21st Annual Symposium on Foundations of Computer Science, pp. 17-27, 1980.
- [MMN02] K. Miura, A. Miyazawa and T. Nishizeki, *Extended rectangular drawing of plane graphs with designated corners*, Proc. Graph Drawing '02, Lect. Notes in Computer Science, Springer, 2528, pp. 256-267, 2002.
- [PS82] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice Hall, Englewood Cliffs, New Jersey, 1982.
- [RNN02] M. S. Rahman, S. Nakano and T. Nishizeki, *Rectangular drawings of plane graphs without designated corners*, *Computational Geometry*, 21, pp. 121-138, 2002.
- [RNN98] M. S. Rahman, S. Nakano and T. Nishizeki, *Rectangular grid drawings of plane graphs*, *Comp. Geom. Theo. Appl.*, 10(3), pp. 203-220, 1998.
- [SY99] S. M. Sait and H. Youssef, *VLSI Physical Design Automation*, World Scientific, Singapore, 1999.
- [SS93] Y. Sun and M. Sarrafzadeh, *Floorplanning by graph dualization: L-shape modules*, *Algorithmica*, 10, pp. 429-456, 1993.
- [Tam87] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, *SIAM J. Comput.*, 16(3), pp. 421-444, 1987.
- [Tho84] C. Thomassen, *Plane representations of graphs*, J. A. Bondy, U. S. R. Murty (Eds.), *Progress in Graph Theory*, Academic Press Canada, Don Mills, Ontario, Canada, pp. 43-69, 1984.
- [Tut54] W. T. Tutte, *A short proof of the factor theorem for finite graphs*, *Canad. J. Math.*, 6, pp. 347-352, 1954.
- [YS93] K. Yeap and M. Sarrafzadeh, *Floor-planning by graph dualization: 2-concave rectilinear modules*, *SIAM J. Comput.*, 22(3), pp. 500-526, 1993.

Approximating the Minmax Subtree Cover Problem in a Cactus

Hiroshi Nagamochi¹ and Taizo Kawada²

¹ Kyoto University,
Yoshida Honmachi, Sakyo, Kyoto 606-8501, Japan
nag@amp.i.kyoto-u.ac.jp

² Toyohashi University of Technology,
Tempaku-cho, Toyohashi 441-8580, Japan
taizo@algo.ics.tut.ac.jp

Abstract. Let $G = (V, E)$ be a connected graph such that edges and vertices are weighted by nonnegative reals. Let p be a positive integer. The minmax subtree cover problem (MSC) asks to find a partition $\mathcal{X} = \{X_1, X_2, \dots, X_p\}$ of V and a set of p subtrees T_1, T_2, \dots, T_p , each T_i containing X_i so as to minimize the maximum cost of the subtrees, where the cost of T_i is defined to be the sum of the weights of edges in T_i and the weights of vertices in X_i . In this paper, we propose an $O(p^2n)$ time $(4 - 4/(p+1))$ -approximation algorithm for the MSC when G is a cactus. This is the first constant factor approximation algorithm for the MSC on a class of non-tree graphs.

1 Introduction

Given a graph, the p -traveling salesmen problem (p -TSP) asks to find a set of p tours that cover all vertices in the graph, minimizing a given objective function. This type of problems arises in numerous applications such as the multi-vehicle scheduling problem [5]. Graphs are restricted to be paths or trees in some applications such as the task sequencing problem, the delivery scheduling by ships on a shoreline [12] and the scheduling of automated guided vehicles. The 1-TSP or p -TSP on paths or trees and analogous routing problems have been studied extensively (e.g., [1, 2, 6, 7, 12]).

Among these problems, this paper considers the minmax subtree cover problem, which is defined in the following.

Let $G = (V, E)$ be an undirected graph, where we may denote the vertex set and the edge set of G by $V(G)$ and $E(G)$, respectively. Let $n = |V(G)|$ and $m = |E(G)|$. We denote by (G, w, h) a graph G such that each edge e and each vertex v are weighted by nonnegative reals $w(e)$ and $h(v)$, respectively. A collection \mathcal{X} of disjoint subsets X_1, X_2, \dots, X_k of V is called a k -partition of V if their union is V , where some X_i may be empty. A collection \mathcal{X} of V is called a p -partition of V if $|\mathcal{X}| = p$. We denote $\sum_{v \in X} h(v)$ for a vertex set X by $h(X)$ and $\sum_{v \in F} w(v)$ for an edge set F by $w(F)$. For a weighted graph (H, w, h) , we may denote $w(E(H))$ by $w(H)$ and $w(H) + h(V(H))$ by $\hat{w}(H)$.

Then the minmax subtree cover problem is described as follows.

Minmax Subtree Cover Problem (MSC):

Input: An instance $I = (G, w, h, p)$ which consists of a connected graph G , an edge-weight w , a vertex-weight h and an integer $p \in [2, n]$.

Feasible solution: A p -partition $\mathcal{X} = \{X_1, X_2, \dots, X_p\}$ of V and a set of p subtrees $\mathcal{T} = \{T_1, T_2, \dots, T_p\}$ of G such that $X_i \subseteq V(T_i)$.

Goal: Minimize $cost(\mathcal{X}, \mathcal{T}) := \max_{1 \leq i \leq p} \{w(T_i) + h(X_i)\}$.

That is, the MSC asks to find a set of p subtrees such that the union of the subtrees covers all vertices in V so as to minimize the maximum cost of the subtrees, where the cost of a subtree T_i is the sum of the weights of edges in T_i and the weights of vertices that are covered by T_i (each vertex is covered by exactly one of the subtrees). Subtrees in \mathcal{T} are not necessarily edge-disjoint or vertex-disjoint. The MSC has an application in the multi-vehicle scheduling problem [6]. The next problem is closely related to the MSC.

Minmax Rooted-Subtree Cover Problem (MRSC):

Input: An instance $I = (G, w, h, r, p)$ which consists of a connected graph G , an edge-weight w , a vertex-weight h , a vertex r designated as a root, and an integer $p \in [2, n]$.

Feasible solution: A p -partition $\mathcal{X} = \{X_1, X_2, \dots, X_p\}$ of V and a set of p subtrees $\mathcal{T} = \{T_1, T_2, \dots, T_p\}$ of G such that $X_i \cup \{r\} \subseteq V(T_i)$.

Goal: Minimize $cost(\mathcal{X}, \mathcal{T}) := \max_{1 \leq i \leq p} \{w(T_i) + h(X_i)\}$.

The MRSC asks to find a set of p subtrees such that each subtree contains r and the union of the subtrees covers all vertices in V , where the objective is to minimize the maximum cost of the subtrees. The MSC and MRSC are both NP-hard. If G is a tree, then there are several approximation algorithms to these problems. For the MSC on a tree G , Averbakh and Berman [4] presented a $(2 - 2/(p+1))$ -approximation algorithm that runs in $O(p^{p-1}n^{p-1})$ time, and Nagamochi and Okada [10] recently gave a polynomial time $(2 - 2/(p+1))$ -approximation algorithm with time complexity $O(p^2n)$.

Averbakh and Berman [3] have given a linear time $4/3$ -approximation algorithm for the MRSC with $p = 2$ on a tree G , and Nagamochi and Okada [10] proposed an $O(n \log \log_{1+\varepsilon/2} 3)$ time $(2 + \varepsilon)$ -approximation algorithm for the MRSC with any integer $p \in [2, n]$ on a tree G . Very recently, Nagamochi [8] proposed a $(3 - 2/(p+1))$ -approximation algorithm for the MRSC on an arbitrary connected graph G .

Note that, for any solution to the MRSC, the set of edges used in the subtrees induces a connected spanning subgraph from G , implying that, for a minimum spanning tree T^* of G , $(w(T^*) + h(V))/p$ is a lower bound on the optimal value to the MRSC. However, no such conventional lower bound is known for the MSC, and no polynomial approximation algorithm has been obtained for the MSC on any class of non-tree graphs so far.

In this paper, we establish a framework for designing approximation algorithms for the MSC on arbitrary graphs, and then give an $O(p^2n)$ time $(4 - 4/(p+1))$ -approximation algorithm for the MSC on a cactus G . This is the first approximation algorithm for the MSC on a class of non-tree graphs

The rest of the paper is organized as follows. Section 2 introduces some terminology and presents preliminary results on the MSC. Section 3 gives a framework of designing approximation algorithms for the MSC. Section 4 analyzes the ratio between lower bounds on trees and cacti in order to design a $(4 - 4/(p + 1))$ -approximation algorithm for the MSC on a cactus. Section 5 describes some concluding remarks.

2 Preliminaries

We denote by (G, w) an edge-weighted graph G such that each edge e is weighted by a nonnegative real $w(e)$, where weight $w(e)$ for an edge $e = (u, v)$ with endvertices $u, v \in V$ may be denoted by $w(u, v)$. A vertex with degree 1 is called a leaf in G , and the set of leaves in G is denoted by $L(G)$. For a subgraph H of (G, w, h) , (H, w) means graph H in which each edge has the same weight in G (i.e., the edge weight of H is the restriction of w on $E(H)$). Similarly we define (H, w, h) for a subgraph H of G . Let $h_{max} = \max_{v \in V} h(v)$.

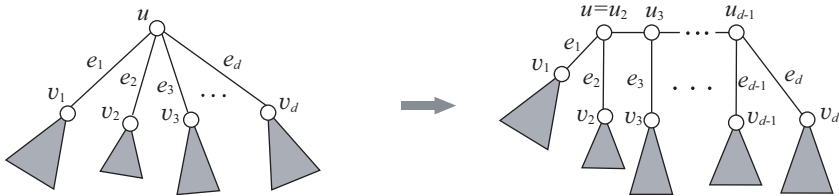


Fig. 1. Examples of cacti

Let T be a tree. For a subset $X \subseteq V(T)$ of vertices, let $T\langle X \rangle$ denote the minimal subtree of T that contains X (where the leaves of $T\langle X \rangle$ will be vertices in X). In this paper, we say that $T\langle X \rangle$ is obtained from T by X . A graph G is called a cactus if no two cycles in G share more than one vertex (see Fig. 1(a)).

We call an instance $I = (G, w, h, p)$ of the MSC a tree instance (resp., cactus instance) if G is a tree (resp., cactus), and denote the optimal value to I by $opt(I)$. We easily observe the following property. For a tree instance $I = (T, w, h, p)$ of the MSC,

$$opt(I) \geq \max \left\{ \frac{\hat{w}(T)}{p}, h_{max} \right\}, \tag{1}$$

provided that there is an optimal solution $(\mathcal{X}, \mathcal{T})$ to I such that each edge is contained in some subtree $T_i \in \mathcal{T}$. In general, (1) does not hold. Nagamochi and Okada [9] have introduced the following lower bound on $opt(I)$. Given an instance $I = (G, w, h, p)$, a p -partition of I is defined to be a set \mathcal{S} of vertex-disjoint subtrees $S_1, S_2, \dots, S_k \subseteq G$ such that each S_i is weighted by a positive integer p_{S_i} with $\sum_{S_i \in \mathcal{S}} p_{S_i} = p$. Define

$$\lambda(\mathcal{S}) = \max \left\{ \frac{\hat{w}(S_i)}{p_{S_i}} \mid S_i \in \mathcal{S} \right\}$$

and

$$\lambda^*(I) = \min\{\lambda(\mathcal{S}) \mid \text{all valued subtree collections } \mathcal{S} \text{ of } I\}.$$

It is a simple matter to see that the next property holds.

Lemma 1. [9] $\dots \dots \dots I \dots \dots \dots, \text{opt}(I) \geq \lambda^*(I)$ □

Nagamochi and Okada [9] have shown the following result.

Theorem 1. [9] $\dots \dots \dots I = (T, w, h, p) \dots \dots \dots$
 $\dots \dots \dots (\mathcal{X}, \mathcal{T}) \dots \dots \dots \text{cost}(\mathcal{X}, \mathcal{T}) \leq \max\{(2 - 2/(p + 1))\lambda^*(I), h_{max}\}$
 $\dots \dots \dots T_i, T_j \in \mathcal{T} \dots \dots \dots$ □

To find such a solution in this theorem in polynomial time, they investigated a relation between the MSC and a problem of minimizing the maximum cost of vertex-disjoint subtrees. Based on the next result, an $O(p^2n)$ time algorithm for constructing a solution in Theorem 1 has been obtained [10].

Theorem 2. [11] $\dots \dots \dots (T = (V, E), w, h) \dots \dots \dots$
 $p \geq 2, \dots \dots \dots F \dots \dots \dots (p - 1) \dots \dots \dots \hat{w}(T')$
 $\dots \dots \dots T' \dots \dots \dots (V, E - F) \dots \dots \dots O(n + \rho p(p + \log \Delta)) \dots \dots \dots \rho$
 $\dots \dots \dots \Delta \dots \dots \dots T, \dots \dots \dots$ □

We remark that there is a cactus instance $I = (G, w, h, p)$ of the MSC such that, for any optimal solution $(\mathcal{X}, \mathcal{T})$, the set of all edges in subtrees in \mathcal{T} does not induce a forest from G (i.e., all edges of some cycle in G are used in \mathcal{T}). Fig. 1(b) shows a cactus instance $I = (G, w, h, p = 4)$ of the MSC such that an optimal solution uses all edges in the cactus, where $h(v) = 0$ for non-leaves v and $h(u_1) = h(u_2) = 30, h(v_1) = h(v_2) = 70,$ and $h(u_3) = h(u_4) = h(v_3) = h(v_4) = 50$ for leaves, and $w(e) = 1$ for all edges. Observe that $\text{opt}(I) = 105$ and an optimal solution consists of four subtrees $T_i (1 \leq i \leq 4)$ such that T_i is a path of five edges connecting u_i and v_i .

To utilize Theorem 1 to approximate the MSC on a cactus G , one may try to transform an optimal solution $(\mathcal{X}, \mathcal{T})$ of a cactus instance $I = (G, w, h, p)$ into an approximate solution of a tree instance $I' = (T, w, h, p)$ for a spanning tree T of G . However, there may be a subtree $T_i \in \mathcal{T}$ which contains an edge e not in T , and transforming T_i by replacing e with other edges in the cycle C containing e may increase the cost of T_i by an arbitrarily large amount compared to its original cost.

For this, we examine the relation between the lower bounds $\lambda^*(I)$ and $\lambda^*(I')$ for cactus instances I and tree instances I' .

Lemma 2. $\dots \dots \dots I = (G = (V, E), w, h, p) \dots \dots \dots G,$
 $\dots \dots \dots T \dots \dots \dots G \dots \dots \dots \lambda^*(I') \leq \lambda^*(I) \dots \dots \dots$
 $\dots \dots \dots I' = (T, w, h, p)$

Let \mathcal{S} be a valued subtree collection in I such that $\lambda(\mathcal{S}) = \lambda^*(I)$. Since G is connected, there is a spanning tree T of G such that $E(T) \supseteq \bigcup_{S \in \mathcal{S}} E(S)$.

Since $\lambda^*(I') \leq \lambda(\mathcal{S})$ holds for the tree instance $I' = (T, w, h, p)$, we have $\lambda^*(I') \leq \lambda(\mathcal{S}) = \lambda^*(I)$ for the tree T . \square

In general, it seems hard to find such a spanning tree T in Lemma 2 without knowing an optimal valued subtree collection \mathcal{S} of I . When G is a tree-like graph, it might be possible to choose a spanning tree T of G such that $\lambda^*(I')$ approximates $\lambda^*(I)$. For an instance $I = (G = (V, E), w, h, p)$, let T be a spanning tree of G , and consider the tree instance $I' = (T = (V, E'), w, h, p)$. Let $\alpha \geq 1$ be a number such that

$$\lambda^*(I') \leq \alpha \cdot \lambda^*(I). \tag{2}$$

In section 4, we shall see that factor α in (2) can be chosen as 2 when G is a cactus.

3 Algorithm for MSC

In this section, we give a framework for designing approximation algorithms for the MSC on arbitrary graphs, based on an approximation algorithm for the MSC on trees [10]. We first convert a given tree instance $I = (T, w, h, p)$ into another tree instance $\tilde{I} = (\tilde{T}, w, h, p)$ by the following procedure.

LOWER_DEGREE

Step 1. For each non-leaf $v \in V(T) - L(T)$, we rename v by v' , set $h(v') := 0$, and add a new leaf, which we now call v , introducing a new edge $e_v = (v', v)$ with $w(e_v) = 0$, where we let the new v have the same weight $h(v)$ as before.

Step 2. For each vertex u with degree $d \geq 4$, execute the following procedure.

Let e_1, e_2, \dots, e_d be the edges incident to u . Split u into $d - 2$ vertices $u_2 (= u), u_3, \dots, u_{d-1}$ introducing new vertices u_3, u_4, \dots, u_{d-1} . Replace the end vertex u of each $e_i, i = 2, 3, \dots, d - 1$ (resp., of e_1 and e_d) with u_i (resp., with u_2 and u_{d-1}). Join the split vertices u_2, u_3, \dots, u_{d-1} by $d - 3$ new edges $(u_2, u_3), \dots, (u_{d-1}, u_d)$ (see Fig. 2). Let weights of all introduced vertices and edges be zero, while the edges e_1, e_2, \dots, e_d have the same weights as before. \square

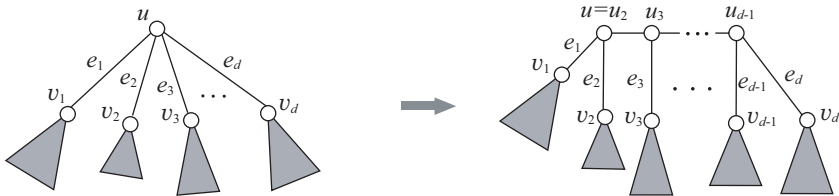


Fig. 2. Splitting a vertex u into $d - 2$ vertices of degree 3

Let $\tilde{I} = (\tilde{T}, w, h, p)$ be the resulting instance, in which every vertex has degree at most three. Then the next two lemmas hold.

Lemma 3. $I = (T, w, h, p)$ $\tilde{I} = (\tilde{T}, w, h, p)$
 LOWER_DEGREE $\lambda^*(\tilde{I}) \leq \lambda^*(I)$

Let $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$ be a valued subtree collection of I with $\lambda(\mathcal{S}) = \lambda^*(I)$, where each S_i is weighted by p_{S_i} . For each $S_i \in \mathcal{S}$, consider the subtree $\tilde{S}_i = \tilde{T}\langle V(S_i) \rangle$ of \tilde{T} . By the construction of \tilde{I} , \tilde{S}_i has cost $w(\tilde{S}_i) + h(V(\tilde{S}_i)) = w(S_i) + h(V(S_i))$. Suppose that there is a vertex $z \in V(\tilde{T}) - V(T)$ that is not covered by any subtree \tilde{S}_i . By construction, such a vertex z satisfies $h(z) = 0$ and is connected to a vertex $u \in V(T)$ in \tilde{T} via a path that consists of edges of weight zero. Thus by attaching such uncovered vertices to some subtrees \tilde{S}_i , we can obtain a valued subtree collection $\tilde{\mathcal{S}}$ of \tilde{I} with $\lambda(\tilde{\mathcal{S}}) = \lambda^*(I)$. This implies $\lambda^*(\tilde{I}) \leq \lambda(\tilde{\mathcal{S}}) = \lambda^*(I)$. \square

Lemma 4. [10] $\tilde{I} = (\tilde{T}, w, h, p)$ $\dots \dots \dots$ $u \dots \dots \dots h(u) = 0$
 $(\tilde{\mathcal{X}}, \tilde{\mathcal{T}}) \dots \dots \dots \tilde{I} \dots \dots \dots \tilde{T}$
 $(\mathcal{X}^*, \mathcal{T}^*) \dots \dots \dots \tilde{I} \dots \dots \dots cost(\mathcal{X}^*, \mathcal{T}^*) \leq cost(\tilde{\mathcal{X}}, \tilde{\mathcal{T}}) \dots \dots \dots \tilde{T}$ \square

Based on the above properties, we obtain the following algorithm for the MSC.

Algorithm APPROX

- Input:** An instance $I = (G, w, h, p)$ of the MSC such that G is connected.
- Output:** A solution $(\mathcal{X}, \mathcal{T})$ to I .
- Step 1.** Find a spanning tree T of G , and let α be a factor such that (2) holds for the tree instance $I' = (T, w, h, p)$.
- Step 2.** Convert the tree instance $I' = (T, w, h, p)$ of the MSC into a tree instance $\tilde{I} = (\tilde{T}, w, h, p)$ by procedure LOWER_DEGREE.
- Step 3.** Find a set of p vertex-disjoint subtrees T_i^* , $i = 1, 2, \dots, p$ in \tilde{I} that minimizes $\max_{1 \leq i \leq p} \hat{w}(T_i^*)$. Let $\mathcal{X}^* = \{X_i^* = V(T_i^*) \mid i = 1, 2, \dots, p\}$.
- Step 4.** Output solution $(\mathcal{X}, \mathcal{T})$ such that $\mathcal{X} = \{X_i = X_i^* \cap V(T) \mid i = 1, 2, \dots, p\}$ and $\mathcal{T} = \{T \langle X_i \rangle \mid i = 1, 2, \dots, p\}$. \square

Theorem 3. $\dots \dots \dots I = (G, w, h, p) \dots \dots \dots$ APPROX $\dots \dots \dots$ $O(\tau_1 + p^2n) \dots \dots \dots (\mathcal{X}, \mathcal{T}) \dots \dots \dots$

$$cost(\mathcal{X}, \mathcal{T}) \leq \alpha \cdot \left(2 - \frac{2}{p+1}\right) opt(I) \tag{3}$$

$\mathcal{T} \dots \dots \dots \tau_1 \dots \dots \dots$

For the instance I' obtained in Step 1, $\lambda^*(I') \leq \alpha \cdot \lambda^*(I) \leq \alpha \cdot opt(I)$ holds by Lemma 1 and condition (2). By Lemma 3, $\lambda^*(\tilde{I}) \leq \lambda^*(I')$ holds for the instance \tilde{I} in Step 2. By Theorem 1, there exists a solution $(\tilde{\mathcal{X}}, \tilde{\mathcal{T}})$ with

$$cost(\tilde{\mathcal{X}}, \tilde{\mathcal{T}}) \leq \max \left\{ \left(2 - \frac{2}{p+1}\right) \lambda^*(\tilde{I}), h_{max} \right\}$$

such that \tilde{T} consists of edge-disjoint subtrees in \tilde{I} . By Lemma 4, instance \tilde{I} has a solution $(\mathcal{X}^*, \mathcal{T}^*)$ such that $cost(\mathcal{X}^*, \mathcal{T}^*) \leq cost(\tilde{\mathcal{X}}, \tilde{T})$ and \mathcal{T}^* consists of vertex-disjoint subtrees in \tilde{I} . Such a solution can be found in Step 3 in $O(p^2n)$ time by using Theorem 2. Note that \mathcal{T} in Step 4 is obtained from \mathcal{T}^* in Step 3 by contracting all edges (of zero) introduced in the construction of \tilde{T} from T . Hence \mathcal{T} consists of edge-disjoint subtrees and satisfies $cost(\mathcal{X}, \mathcal{T}) = cost(\mathcal{X}^*, \mathcal{T}^*)$. Therefore, from the above inequalities and $opt(I) \geq h_{max}$, we have (3). We easily see that APPROX can be implemented to run in $O(\tau_1 + p^2n)$ time. \square

4 Factor α for Cacti

In this section, we show that every cactus G contains a spanning tree T such that (2) holds for factor $\alpha = 2$.

Lemma 5. $I = (G, w, h, p) \dots G \dots (T, w)$
 $2\lambda^*(I) \dots (G, w) \dots \lambda^*(I) \leq \dots I' = (T, w, h, p) \dots \square$

This lemma and Theorem 3 imply the next result.

Corollary 1. $I = (G, w, h, p) \dots G \dots$
 $(\mathcal{X}, \mathcal{T}) \dots cost(\mathcal{X}, \mathcal{T}) \leq (4 - 4/(p+1))opt(I) \dots$
 $O(p^2n) \dots \square$

In the rest of this section, we prove Lemma 5. Let $\lambda^* = \lambda^*(I)$, \mathcal{S} be a valued subtree collection to $I = (G, w, h, p)$ with $\lambda(\mathcal{S}) = \lambda^*$, and (T, w) be a minimum spanning tree of (G, w) . For notational simplicity, we assume that a given cactus G has no bridge, i.e., G consists only of cycles (if necessary, we add to each bridge $e = (u, v)$ a new edge $e' = (u, v)$ with $w(e') = w(e)$, and we can assume that no new edge is included in any subtree in \mathcal{S}).

Let $\bar{E} = E(G) - E(T)$, where, by the minimality of T , each edge $e \in \bar{E}$ has the maximum edge weight among edges in the cycle containing e . Let $E(\mathcal{S}) = \cup_{S \in \mathcal{S}} E(S)$. Notice that \mathcal{S} may not be a valued subtree collection of $I' = (T, w, h, p)$, i.e., some edges in \bar{E} may be used in subtrees in \mathcal{S} .

We now give an algorithm for transforming \mathcal{S} into a valued subtree collection \mathcal{S}' of I' . The algorithm consists of two phases. In the first phase, we break a subtree $S \in \mathcal{S}$ into two (or into a smaller subtree S and a fraction), where a fraction is a triplet (E', V', p') of an edge set $E' \subseteq E(S)$, a vertex set $V' \subseteq V(S)$ and an integer $p' \in [0, p_S]$. For a fraction $\delta = (E', V', p')$, define $\hat{w}(\delta) = w(E') + w(V')$ and $p(\delta) = p'$, respectively. During the first phase, we maintain a set \mathcal{S} of vertex-disjoint subtrees S , each weighted by a positive integer p_S such that

$$\cup_{S \in \mathcal{S}} V(S) \subseteq V, \quad \sum_{S \in \mathcal{S}} p_S \leq p, \quad \lambda(\mathcal{S}) \leq \lambda^*. \tag{4}$$

The resulting \mathcal{S} after the first phase is not a valued subtree collection to I' if $\cup_{S \in \mathcal{S}} V(S) \neq V$ or $\sum_{S \in \mathcal{S}} p_S < p$. In the second phase, we modify subtrees $S_i \in \mathcal{S}$

by attaching fractions δ to them so that the set of the resulting subtrees becomes a valued subtree collection \mathcal{S}' of I' and $\lambda(\mathcal{S}') \leq 2\lambda^*$ holds.

4.1 Phase-1

We now describe a procedure for the first phase. For each subtree $S_i \in \mathcal{S}$, let $\Delta_i := \emptyset$, where Δ_i stores a set of fractions δ that will be added to S_i in the second phase.

Choose a cycle C_0 in G as a root cycle, and define distance $dist(v)$ for a vertex $v \in V$ to be the number of cycles which share edges with a simple path from a vertex in C_0 to v in G . For each cycle C , define $dist(C) = \min_{v \in V(C)} dist(v)$ and call the vertex $v \in V(C)$ with $dist(v) = dist(C)$ the *parent* of C . Then we number all cycles as C_0, C_1, \dots, C_r such that $dist(C_i) \leq dist(C_j)$ for any $i < j$.

For each cycle $C = C_r, C_{r-1}, \dots, C_1$ in this order, we apply the following procedure CUT if there is a subtree $S_i \in \mathcal{S}$ with $E(S_i) \cap \bar{E} \cap E(C) \neq \emptyset$; we skip cycle C otherwise.

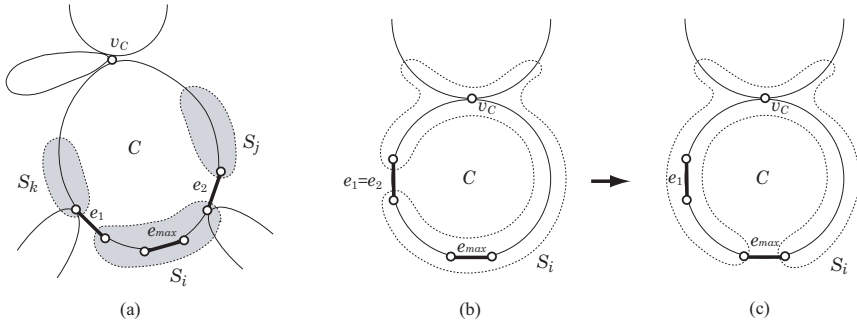


Fig. 3. (a) Illustration for subtrees $S_i, S_j, S_k \in \mathcal{S}$ and edges $e_{max}, e_1, e_2 \in E(C)$, (b) and (c) Illustrations for Case-1 with $e_1 = e_2$

Procedure CUT Let e_{max} be the edge in $E(S_i) \cap \bar{E} \cap E(C)$, and $v_C \in V(C)$ be the parent of C . Let $e_1, e_2 \in E(C) - E(S_i)$ be the edges incident to S_i (possibly $e_1 = e_2$), and S_k (resp., S_j) be the subtree that is adjacent to S_i via edge e_1 (resp., e_2) (possibly $S_k = S_j$). See Fig. 3(a).

Case-1: $e_1 = e_2$ (hence $V(S_i) = V(C)$) (Fig. 3(b)). Let S_i be the subtree obtained from S_i by replacing e_{max} with e_1 (Fig. 3(c)).

Case-2: S_j or S_k does not contain $v_C \in V(S_i)$ (assume $v_C \notin V(S_k)$ without loss of generality) (Fig. 4(a)). Let S'_i and S''_i be two subtrees obtained from S_i by removing edge e_{max} ; S'_i is assumed to be adjacent to S_k without loss of generality. Let

$$p_{S'_i} := \left\lceil \frac{\hat{w}(S'_i)}{\hat{w}(S_i)} p_{S_i} \right\rceil, \quad p_{S''_i} := p_{S_i} - \left\lceil \frac{\hat{w}(S'_i)}{\hat{w}(S_i)} p_{S_i} \right\rceil,$$

$$\delta := (V(S''_i), E(S''_i) \cup \{e_1\}, p_{S''_i}), \quad \Delta_k := \Delta_k \cup \{\delta\},$$

and

$$S_i := S'_i, \quad p_{S_i} := p_{S'_i}$$

(see Fig. 4(b)).

Case-3: $S_j = S_k$ and $v_C \in V(S_j)$ (Fig. 4(c)). Let

$$\delta := (V(S_i), (E(S_i) - \{e_{max}\}) \cup \{e_1, e_2\}, p_{S_i}), \quad \Delta_j := \Delta_j \cup \{\delta\}$$

and

$$S := S - \{S_i\}, \quad \Delta_j := \Delta_j \cup \Delta_i$$

(see Fig. 4(d)).

□

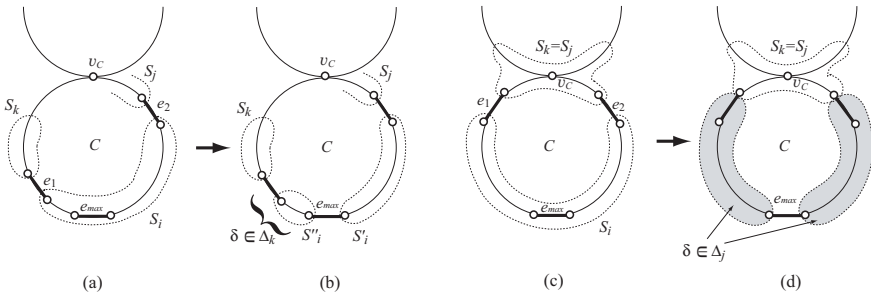


Fig. 4. (a),(b) Illustrations for Case-2; (c), (d) Illustrations for Case-3

Claim 1 S (4)

In Cases-3, subtree S_i is simply removed from the current S . In other cases, S_i and p_{S_i} will be modified. In Case-1, p_{S_i} remains unchanged, while $\hat{w}(S_i)$ never increases since $w(e_{max}) \geq w(e_1)$. In Cases-2, subtree S_i is modified into subtree S'_i such that

$$\frac{\hat{w}(S'_i)}{p_{S'_i}} = \frac{\hat{w}(S'_i)}{\lceil \frac{\hat{w}(S'_i)}{\hat{w}(S_i)} p \rceil} \leq \frac{\hat{w}(S'_i)}{\frac{\hat{w}(S'_i)}{\hat{w}(S_i)} p_{S_i}} = \frac{\hat{w}(S_i)}{p_{S_i}},$$

which preserves property $\lambda(S) \leq \lambda^*$. We easily observe that the first two conditions in (4) remain valid after each application of CUT. □

Claim 2 δ $\hat{w}(\delta)/p(\delta) \leq \lambda^*$.
 $p(\delta) \geq 1$ $\hat{w}(\delta) < \lambda^*$. $p(\delta) = 0$

By Claim 1, any subtree $S \in \mathcal{S}$ after each application of CUT satisfies $p_S \geq 1$ and $\hat{w}(S)/p_S \leq \lambda^*$. Then a fraction $\delta = (V(S_i), (E(S_i) - \{e_{max}\}) \cup \{e_1, e_2\}, p_{S_i})$ in Case-3 satisfies

$$\frac{\hat{w}(\delta)}{p(\delta)} = \frac{\hat{w}(S_i) - w(e_{max}) + w(e_1) + w(e_2)}{p_{S_i}} \leq \frac{\hat{w}(S_i) + w(e_2)}{p_{S_i}} \leq 2 \frac{\hat{w}(S_i)}{p_{S_i}} \leq 2\lambda^*.$$

We next consider a fraction $\delta = (V(S''_i), E(S''_i) \cup \{e_1\}, p_{S''_i})$ in Case-2. Note that $p_{S''_i} = p_{S_i} - \lfloor p_{S_i} \hat{w}(S''_i) / \hat{w}(S_i) \rfloor = \lfloor p_{S_i} (\hat{w}(S''_i) + w(e_{max})) / \hat{w}(S_i) \rfloor$. If $p_{S''_i} \geq 1$, then it holds

$$\frac{\hat{w}(\delta)}{p(\delta)} = \frac{\hat{w}(S''_i) + w(e_1)}{\lfloor \frac{\hat{w}(S''_i) + w(e_{max})}{\hat{w}(S_i)} p_{S_i} \rfloor} \leq 2 \cdot \frac{\hat{w}(S''_i) + w(e_{max})}{\frac{\hat{w}(S''_i) + w(e_{max})}{\hat{w}(S_i)} p_{S_i}} = 2 \cdot \frac{\hat{w}(S_i)}{p_{S_i}} \leq 2\lambda^*.$$

On the other hand, if $p_{S''_i} < 1$, i.e., $p_{S_i} (\hat{w}(S''_i) + w(e_{max})) / \hat{w}(S_i) < 1$, then we have

$$\hat{w}(\delta) = \hat{w}(S''_i) + w(e_{max}) < \frac{\hat{w}(S_i)}{p_{S_i}} \leq \lambda^*.$$

This proves the claim. □

Claim 3 $\dots S_i \in \mathcal{S} \dots \Delta_i \dots p(\delta) = 0$

Any fraction δ with $p(\delta) = 0$ is created in Case-2 when a cycle C is being scanned, and added to Δ_k of a subtree S_k with $V(S_k) \cap V(C) \neq \emptyset$ and $v_C \notin V(S_k)$. This implies that S_k cannot receive any other fraction δ' with $p(\delta') = 0$. Therefore, Δ_i contains at most one fraction δ with $p(\delta) = 0$. □

4.2 Phase-2

In the second phase, we paste fractions in Δ_i to the subtree $S_i \in \mathcal{S}$ by the following procedure.

Procedure PASTE

Let \mathcal{S} be the set of subtrees obtained in Phase-1. For each subtree $S_i \in \mathcal{S}$ with $\Delta_i \neq \emptyset$, we modify S_i as follows: For $\Delta_i = \{(V'_j, E'_j, p'_j) \mid j = 1, 2, \dots, k\}$, let

$$S_i^* := (V(S_i) \cup \bigcup_{1 \leq j \leq k} V'_j, E(S_i) \cup \bigcup_{1 \leq j \leq k} E'_j)$$

and

$$p_{S_i^*} := p_{S_i} + \sum_{1 \leq j \leq k} p'_j.$$

Let \mathcal{S}^* be the set of resulting subtrees. □

By construction of fractions in CUT, we easily see that S_i computed by PASTE is a subtree and that the resulting set \mathcal{S}^* is a set of vertex-disjoint subtrees of T such that $\cup_{S \in \mathcal{S}^*} V(S) = V$ and $\sum_{S \in \mathcal{S}^*} p_S = p$, i.e., \mathcal{S}^* is a valued subtree collection of $I' = (T, w, h, p)$.

Lemma 6. $\dots \mathcal{S}^* \dots I' = (T, w, h, p) \dots \lambda(\mathcal{S}^*) \leq 2\lambda^*$

Let \mathcal{S} be a set of subtrees obtained after Phase-1. Consider a subtree $S_i^* \in \mathcal{S}^*$, and let $\Delta_i = \{\delta_1, \delta_2, \dots, \delta_k\}$, where $\hat{w}(S_i^*) = \hat{w}(S_i) + \hat{w}(\delta_1) + \dots + \hat{w}(\delta_k)$ and $p_{S_i^*} = p_{S_i} + p(\delta_1) + \dots + p(\delta_k)$ hold for $S_i \in \mathcal{S}$. Then it suffices to show that

$$\frac{\hat{w}(S_i^*)}{p_{S_i^*}} = \frac{\hat{w}(S_i) + \hat{w}(\delta_1) + \dots + \hat{w}(\delta_k)}{p_{S_i} + p(\delta_1) + \dots + p(\delta_k)} \leq 2\lambda^*. \tag{5}$$

If $p(\delta_j) \neq 0$ for all $\delta_j \in \Delta_i$, then it holds $[\hat{w}(S_i) + \hat{w}(\delta_1) + \dots + \hat{w}(\delta_k)]/[p_{S_i} + p(\delta_1) + \dots + p(\delta_k)] \leq \max\{\hat{w}(S_i)/p_{S_i}, \hat{w}(\delta_1)/p(\delta_1), \dots, \hat{w}(\delta_k)/p(\delta_k)\} \leq 2\lambda^*$ by Claim 2. We now consider the case where Δ_i contains a fraction δ with $p(\delta) = 0$. By Claim 3, Δ_i contains exactly one such fraction, say δ_1 with $p(\delta_1) = 0$. Therefore, by Claim 2, we have

$$\begin{aligned} \frac{\hat{w}(S_i^*)}{p_{S_i^*}} &\leq \max\left\{\frac{\hat{w}(S_i) + \hat{w}(\delta_1)}{p_{S_i}}, \frac{\hat{w}(\delta_2)}{p(\delta_2)}, \dots, \frac{\hat{w}(\delta_k)}{p(\delta_k)}\right\} \\ &\leq \max\left\{\frac{\hat{w}(S_i) + \lambda^*}{p_{S_i}}, 2\lambda^*\right\} \leq 2\lambda^*, \end{aligned}$$

as required. □

This completes the proof of Lemma 5.

5 Concluding Remarks

In this paper, we have designed a framework for designing approximation algorithms for the MSC on an arbitrary graph, and have given an $O(p^2n)$ time $(4 - 4/(p+1))$ -approximation algorithm for the MSC on a cactus. Our framework for designing algorithms for the MSC seems effective on classes of graphs which have a similar structure with trees. It would be interesting to investigate factors α in (2) for those classes such as outerplanar graphs.

References

1. T. Asano, N. Katoh and K. Kawashima, A new approximation algorithm for the capacitated vehicle routing problem on a tree, *J. Combinatorial Optimization*, 5 (2001) 213–231.
2. I. Averbakh and O. Berman, Sales-delivery man problems on treelike networks, *Networks*, 25 (1995) 45–58.
3. I. Averbakh and O. Berman, A heuristic with worst-case analysis for minmax routing of two traveling salesmen on a tree, *Discrete Applied Mathematics*, 68 (1996) 17–32.
4. I. Averbakh and O. Berman, $(p-1)/(p+1)$ -approximate algorithm for p -traveling salesmen problems on a tree with minmax objective, *Discrete Applied Mathematics*, 75 (1997) 201–216.
5. J. Desrosiers, Y. Dumas, M. M. Solomon and F. Soumis, Time constrained routing and scheduling, In M. O. Ball, T. L. Magnanti, C. L. Monma and G. L. Nemhauser (eds.): *Handbooks in Operations Research and Management Science Volume 8: Network Routing* (North-Holland, 1995), 35–139.

6. Y. Karuno and H. Nagamochi, 2-approximation algorithms for the multi-vehicle scheduling on a path with release and handling times, *Discrete Applied Mathematics*, 129 (2003) 433–447.
7. Y. Karuno and H. Nagamochi, An approximability result of the multi-vehicle scheduling problem on a path with release and handling times, *Theoretical Computer Science A*, 312 (2004) 267–280.
8. H. Nagamochi, Approximating the minmax rooted-subtree cover problem (submitted to a journal).
9. H. Nagamochi and K. Okada, A faster 2-approximation algorithm for the minmax p -traveling salesmen problem on a tree, *Discrete Applied Mathematics*, 140 (2004) 103–114.
10. H. Nagamochi and K. Okada, Polynomial time 2-approximation algorithms for the minmax subtree cover problem, *Lecture Notes in Computer Science*, 2906, Springer-Verlag, (2003) 138–147.
11. Y. Perl and U. Vishkin, Efficient implementation of a shifting algorithm, technique for the partitioning, *Discrete Applied Mathematics*, 12 (1985) 71–80.
12. H. Psaraftis, M. Solomon, T. Magnanti and T. Kim, Routing and scheduling on a shoreline with release times, *Management Science*, 36 (1990) 212–223.

Boundary-Optimal Triangulation Flooding*

Richard J. Nowakowski¹ and Norbert Zeh²

¹ Department of Mathematics and Statistics, Dalhousie University, Halifax,
NS B3H 3J5, Canada
rjn@mathstat.dal.ca

² Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 1W5, Canada
nzeh@cs.dal.ca

Abstract. Given a planar triangulation all of whose faces are initially white, we study the problem of colouring the faces black one by one so that the boundary between black and white faces as well as the number of connected black and white regions are small at all times. We call such a colouring sequence of the triangles a flooding. Our main result shows that it is in general impossible to guarantee boundary size $\mathcal{O}(n^{1-\epsilon})$, for any $\epsilon > 0$, and a number of regions that is $o(\log n)$, where n is the number of faces of the triangulation. We also show that a flooding with boundary size $\mathcal{O}(\sqrt{n})$ and $\mathcal{O}(\log n)$ regions can be computed in $\mathcal{O}(n \log n)$ time.

1 Introduction

We study the following problem posed by Hurtado [8]: Given a planar triangulation all of whose triangles are initially white, find an ordering of the triangles such that, when colouring the triangles black in this order, the number of connected monochromatic regions and the boundary between white and black regions are small at all times.

In this paper, we provide an $\mathcal{O}(n \log n)$ -time algorithm that finds an order of colouring the triangles such that, at all times, the boundary between the black and the white regions has size $\mathcal{O}(\sqrt{n})$, there are $\mathcal{O}(\log n)$ white regions, and there is only one black region. We also show that the boundary size as well as the number of white regions are best possible up to constant factors. In particular, there cannot be any floodings with boundary size $o(\sqrt{n})$ in general because, at the time when half the faces are black and half the faces are white, we would have a $\frac{1}{2}$ -separator of size $o(\sqrt{n})$; but Lipton and Tarjan [11] show that the minimal size of such a separator is $\Omega(\sqrt{n})$ in the worst case. As for the number of white regions, we show that, for any $\epsilon > 0$, there exists a family of triangulations such that, if a boundary size of $\mathcal{O}(n^{1-\epsilon})$ is desired, there have to be $\Omega(\log n)$ monochromatic regions at some point during the colouring process.

The triangulation flooding problem can be rephrased as a layout problem of the dual of the triangulation: Given a 3-regular planar graph, find a linear layout (v_1, v_2, \dots, v_n) of small cutwidth and such that for every cut $(V_i, V \setminus V_i)$, the

* This work was partially supported by NSERC.

number of connected components of $G - E_i$ is small, where $V_i = \{v_1, v_2, \dots, v_i\}$ and E_i is the set of edges with exactly one endpoint in V_i ; that is, the edges in E_i cross the cut.

Layouts of small cutwidth have applications in network reliability [9], graph drawing [15], and rendering and stream processing of triangular meshes [1, 2, 7]. The cutwidth of a graph is also closely related to its search number, which is relevant in VLSI design [10] and network security [3, 6, 16]. See [4] for a comprehensive survey of graph layout problems and their applications. Next we discuss some results in this area that are closely related to our work.

It is well-known that every planar graph of maximal degree d has cutwidth $\mathcal{O}(\sqrt{dn})$ [5]; for 3-regular planar graphs, this implies that the cutwidth is $\mathcal{O}(\sqrt{n})$. Our result shows that 3-regular planar graphs have linear layouts of cutwidth $\mathcal{O}(\sqrt{n})$ that are “well-behaved” in the sense that their cuts do not partition the graph into too many connected components.

In [2], the following model for rendering a triangular mesh is studied: Vertices are pushed onto (and popped from) a vertex stack. To render a triangle, it is necessary that all three vertices of the triangle are on the stack. The question studied in [2] is to determine the stack size required to render any triangular mesh while pushing every vertex onto the stack only once. It is shown that a stack of size $\mathcal{O}(\sqrt{n})$ suffices. In order to obtain this result, a recursive separator decomposition similar to the one used in [5] for computing a layout of small cutwidth is used. Our approach for finding a colouring sequence of the triangles also uses a recursive separator decomposition. The main difference is that we use simple-cycle separators [14] in our partition and that we combine these separators carefully to keep the number of monochromatic regions small.

While keeping the number of monochromatic regions small at all times is of limited importance for mesh rendering (apart from aesthetic concerns if the rendering process is slow), it is important for encoding triangular meshes in a minimal number of bits [1, 7], which is desirable for applications that need to stream triangular meshes over communication channels of limited bandwidth such as the internet. The encoding schemes of [1, 7] achieve a compression of between 0.2 and 3.5 bits per vertex, depending on the regularity of the mesh. Similar to our triangulation flooding problem, the idea is to start at a triangle of the mesh and encode the remaining triangles one at a time so that the next triangle to be encoded is chosen from among the triangles that share an edge with a previously encoded triangle. If the third vertex of the triangle is a new vertex, the degree of the vertex is encoded. If the vertex has been seen before, the addition of the triangle may split the set of triangles yet to be encoded into two regions. This needs to be recorded using a so-called split code whose length is logarithmic in the length of the current boundary cycle. Thus, split codes are much more expensive than the encoding of new vertices and should be avoided. Using our approach, the boundary size is guaranteed to be $\mathcal{O}(\sqrt{n})$, which helps to bound the length of the split codes. We also guarantee that the number of lists on the boundary list stack in the algorithm of [1] is $\mathcal{O}(\log n)$; each list on this stack represents a boundary cycle of the already encoded region. Unfortunately,

the total number of split operations may still be linear and, other than in the algorithm of [1], the order in which the triangles are encoded cannot easily be determined from the degrees of the boundary vertices and, thus, needs to be explicitly encoded.

Finally, we want to point out the relationship between our results and the graph search problem where a number of searchers try to clean the edges of a graph all of whose edges are contaminated. In order to clean an edge, a searcher has to move along the edge; if there exists a path between a contaminated edge and a clean edge that is free of searchers, the edge is recontaminated. The search number of a graph is the minimal number of searchers that are sufficient to clean the graph. The connected search number is the minimal number of searchers that are sufficient to clean the graph and, in addition, guarantee that the graph spanned by clean edges is connected at all times. Determining the search number of a graph is NP-complete [13]. For graphs of maximal degree 3, it is known that the search number is equal to the graph's cutwidth [12]. Using the result of [5], this implies that the search number of a 3-regular planar graph is $\mathcal{O}(\sqrt{n})$. In fact, the proof of [5] immediately implies that the search number of a 3-regular planar graph is $\mathcal{O}(\sqrt{n})$. Our result proves that, for 3-regular planar graphs, $\mathcal{O}(\sqrt{n})$ searchers suffice to clean the graph while keeping the clean subgraph connected and keeping the number of maximal connected contaminated subgraphs small.

2 Preliminaries

In this section, we introduce the terminology used in this paper and discuss previous results that are used in our algorithms.

Let T be a given planar triangulation with n faces (triangles). A colouring c of T is an assignment of a colour $c(f) \in \{\text{black}, \text{white}\}$ to every face f of T . Colouring c is *trivial* if it colours all faces black or all faces white. The boundary ∂c of a non-trivial colouring c is the set of edges in T whose two adjacent triangles have different colours. We say that a colouring c is $t(n)$ -bounded if $|\partial c| \leq t(n)$. Consider the connected components of $\mathbb{R}^2 \setminus \partial c$. Each of these components contains either only black or only white faces. Accordingly, we call such a region black or white. We define W and B to be the union of all white and black faces, respectively. We say that colouring c is $(b(n), w(n))$ -scattered if B consists of at most $b(n)$ connected regions and W consists of at most $w(n)$ connected regions. The colouring is $s(n)$ -scattered if the total number of monochromatic regions is at most $s(n)$; that is, a $(b(n), w(n))$ -scattered colouring is also $(b(n) + w(n))$ -scattered. The black count of a colouring c is the number of faces f of T such that $c(f) = \text{black}$. A flooding F is a sequence $F = (c_0, c_1, \dots, c_n)$ of colourings such that, for all $0 \leq i \leq n$, the black count of colouring c_i is i and $c_i(f) = \text{black}$ implies that $c_{i+1}(f) = \text{black}$. We say that a flooding F is $t(n)$ -bounded, $(b(n), w(n))$ -scattered, or $s(n)$ -scattered if every non-trivial colouring in F is $t(n)$ -bounded, $(b(n), w(n))$ -scattered, or $s(n)$ -scattered, respectively.

If both regions W and B are connected, the boundary of colouring c is a simple cycle C . We call this cycle a $\sqrt{8n}$ -boundary of T . Given an assignment of weights to the faces of T such that the weights of all faces sum to 1, we call cycle C a $\frac{2}{3}$ - ϵ -separator if the total weight of the faces in each of the regions W and B is at most ϵ . The $\frac{2}{3}$ -weight of a simple-cycle separator is the number of edges on the cycle. Miller shows the following result.

Theorem 1 (Miller [14]). *Let T be a planar triangulation with n faces. Then there exists a $\frac{2}{3}$ - ϵ -separator of T with $\frac{2}{3}$ -weight $\leq \sqrt{8n}$.*

3 New Results

In Section 4, we prove that every planar triangulation T has an $\mathcal{O}(\sqrt{n})$ -bounded $(1, \mathcal{O}(\log n))$ -scattered flooding. More strongly, given a face f_0 of T , there exists such a flooding $F = (c_0, c_1, \dots, c_n)$ such that $c_1(f_0) = \text{black}$; that is, f_0 is the first triangle coloured black and every subsequent triangle is coloured black only after at least one of its neighbours has been coloured black. We show that such a flooding can be computed in $\mathcal{O}(n \log n)$ time.

In Section 5, we prove that the result from Section 4 is best possible in a very strong sense: For any $0 < \epsilon < 1$, there exists a family of triangulations that do not have $\mathcal{O}(n^{1-\epsilon})$ -bounded $o(\log n)$ -scattered floodings.

4 Floodings with Small Boundary and Low Scatter

In this section, we show how to efficiently find a flooding for a given triangulation that maintains a small boundary at all times, keeps the black region connected, and creates only few white regions at any time. The following theorem states this formally:

Theorem 2. *Let T be a planar triangulation with n faces. Then for any face f_0 of T , there exists a $(1, \mathcal{O}(\log n))$ -scattered flooding $F = (c_0, c_1, \dots, c_n)$ such that $c_1(f_0) = \text{black}$ and the boundary of F has $\mathcal{O}(n \log n)$ edges.*

To prove Theorem 2, we first compute a recursive partition \mathcal{P} of T using simple-cycle and multi-path separators and then use an ordering of the leaves of \mathcal{P} to compute the order in which to colour the faces of T . The subgraphs of T in this partition are, as defined as follows: A partial triangulation is an embedded planar graph G with the following properties:

- (i) Every face of G is marked as either interior or exterior.
- (ii) All interior faces are triangles.
- (iii) The union of all interior faces forms a connected region.
- (iv) No two exterior faces share an edge.

The boundary ∂G of G is the set of edges on the boundaries of the exterior faces. To compute the partition \mathcal{P} , we make use of the following lemma.

Lemma 1. Let G be a partial triangulation with $n \geq 2$ interior faces.

- (i) If C is a simple cycle of G with length at most $\sqrt{8n + 16}$, then C partitions G into regions of size at most $2n/3$.
- (ii) If P_1, P_2, \dots, P_k are maximal sub-paths of C that consist of only interior edges of G , then C partitions G into regions of size at most $2n/3$.

We triangulate the exterior faces of G and give weight $1/n$ to every interior face of G and weight 0 to every triangle produced by triangulating the exterior faces. We use Theorem 1 to find a simple-cycle separator C of the resulting triangulation T . Since the interior of T is connected, T has at most $n + 2$ vertices. Hence, by Theorem 1, the cycle C has length at most $\sqrt{8n + 16}$. If C contains only interior edges of G , we have Case (i). Otherwise, let P_1, P_2, \dots, P_k be the maximal sub-paths of C that consist of only interior edges of G . Every path P_i is simple because C is simple. Every region in the partition of G is completely on one side of C and thus has size at most $2n/3$. Thus, we have Case (ii). The complexity of all three steps of this procedure is linear. \square

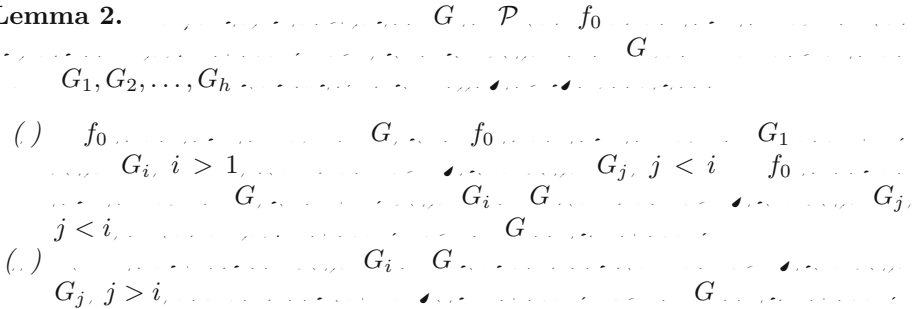
Given a partial triangulation G , a separator as in Lemma 1 partitions G into partial triangulations G_1, G_2, \dots, G_h as follows: Let R_1, R_2, \dots, R_h be the connected regions into which the interior of G is divided by the separator given by the lemma. Then the partial triangulation G_i has the faces of R_i as its interior faces. The exterior faces of G_i are bounded by the boundary edges of R_i . The following observation is an immediate consequence of the simplicity of C .

Observation 1. Let G be a partial triangulation with n interior faces. Then the partial triangulations G_1, G_2, \dots, G_h have at most $2n/3$ interior faces each.

Next we define a recursive partition \mathcal{P} of T into partial triangulations by repeated application of Lemma 1. The root of \mathcal{P} is T . For every non-leaf partial triangulation G , its children are obtained by applying Lemma 1 to G . Every non-leaf partial triangulation has more than \sqrt{n} interior faces; every leaf partial triangulation has at most \sqrt{n} interior faces. To construct the desired flooding F , we compute an ordering of the leaf partial triangulations of \mathcal{P} and flood each triangulation in turn. We obtain this ordering as the left-to-right ordering of the leaves of \mathcal{P} obtained by ordering the children of every internal node of \mathcal{P} , from the root toward the leaves. Given that the children of the ancestors of a partial triangulation G in \mathcal{P} have already been ordered, we give colours black and white to the boundary edges of G as follows: Let e be such an edge, let f be the face in G incident to e , let f' be the other face incident to e , and let G' be the leaf

triangulation that contains f' . Let H and H' be the ancestors of G and G' that are children of the LCA H'' of G and G' . If H' precedes H in the order of the children of H'' , edge e is black; otherwise, edge e is white. To order the children of G , we now apply the following lemma.

Lemma 2.



We call a boundary edge of a child G_i of G black, white, or internal depending on whether it is a black or white boundary edge of G or an edge between two children of G . We partition the children of G into three groups: If f_0 is an interior face of G , then \mathcal{G}_2 contains the child G_1 that contains f_0 and all remaining faces are in \mathcal{G}_3 . \mathcal{G}_1 is empty in this case. If f_0 is not an interior face of G , then \mathcal{G}_3 contains all children whose boundary edges are white or internal. \mathcal{G}_2 contains all children that have at least one black boundary edge and at least one boundary edge that is white or is shared with a child in \mathcal{G}_3 . Group \mathcal{G}_1 contains the remaining children, that is, those whose boundary edges are either black or shared with other children in \mathcal{G}_1 and \mathcal{G}_2 . If groups \mathcal{G}_2 and \mathcal{G}_3 are empty, we move an arbitrary child from group \mathcal{G}_1 to group \mathcal{G}_2 .

In the ordering of the children of G , the children in \mathcal{G}_1 precede the children in \mathcal{G}_2 , which in turn precede the children in \mathcal{G}_3 . The children in \mathcal{G}_2 are arranged in no particular order. The children in \mathcal{G}_3 are arranged so that each such child G_i shares an edge with a child $G_j, j < i$. Since $\mathcal{G}_1 \cup \mathcal{G}_2$ is non-empty, the connectivity of the interior of G implies the existence of such an ordering. The children in \mathcal{G}_1 are arranged so that each such child G_i shares an edge with a child $G_j, j > i$. Again, the non-emptiness of $\mathcal{G}_2 \cup \mathcal{G}_3$ and the connectivity of the interior of G imply the existence of such an ordering.

If f_0 is in G , then Condition (i) is trivially satisfied by the constructed ordering. So assume that f_0 is not in G . Then every face in $\mathcal{G}_1 \cup \mathcal{G}_2$ has a black boundary edge and every child G_i in \mathcal{G}_3 shares an edge with a child $G_j, j < i$. To see that Condition (ii) is satisfied, observe that, by Observation 1, at most one child in \mathcal{G}_3 does not have a white boundary edge. Every child G_i in \mathcal{G}_1 shares an edge with a child $G_j, j > i$; every child in \mathcal{G}_2 shares an edge with a child G_j in \mathcal{G}_3 or has a white boundary edge. □

Given the ordering of the leaf partial triangulations of \mathcal{P} produced by top-down application of Lemma 2, we flood these partial triangulations in this order. The first leaf partial triangulation contains f_0 and is flooded starting from f_0 . Every subsequent triangulation G is flooded starting from a face that has a black

edge of G on its boundary. A simple inductive argument shows that such a face always exists. (Details appear in the full paper.) The following lemma shows how to flood leaf triangulations.

Lemma 3. $\dots G \dots f \dots G$
 $\sqrt{n} + 2, \dots G \dots \log \sqrt{n}$

We keep the black region connected by colouring f black and subsequently colouring a face f' black only if at least one of its adjacent faces is black. The bound on the boundary size follows immediately from the fact that G has at most \sqrt{n} interior faces. To guarantee that there are never more than $\log \sqrt{n}$ white regions, we choose the order in which to colour triangles using the following recursive procedure: Let R be the current region to be coloured. After colouring f black, R is the interior of G minus f . We choose a face f' in R adjacent to a black face and colour it black. Face f' divides R into at most two connected regions. We flood each of them recursively, first the smaller one, then the bigger one. If we consider this recursive partition of the interior of G into white regions, then at any time only $\log \sqrt{n}$ ancestors of the current region can have siblings waiting to be flooded because each such sibling is of at least the same size as the corresponding ancestor of the current region. Every such sibling is completely contained in a white region of the current colouring of G , and every white region consists of at least one such sibling. Hence, there are at most $\log \sqrt{n}$ white regions at any time. \square

The next three lemmas establish that the computed flooding F has the desired properties.

Lemma 4. $\dots F \dots$

This is obvious, once we observe that F floods the children of any partial triangulation in \mathcal{P} in left-to-right order. Hence, for every leaf partial triangulation G , a black boundary edge is one that already has an incident black triangle at the time when G is being flooded. The flooding of G starts at a triangle incident to such an edge and keeps the black region connected, by Lemma 3. \square

Lemma 5. $\dots F \dots \mathcal{O}(\sqrt{n})$

Consider the leaf partial triangulation G currently being flooded. Only ancestors of G can be bichromatic. Thus, the boundary of the black region is part of the separators computed at these ancestors. Since the separator of an ancestor with n' internal faces has size $\mathcal{O}(\sqrt{n'})$ and the sizes of these ancestors are geometrically decreasing, the proper ancestors of G contribute $\mathcal{O}(\sqrt{n})$ to the boundary size of the colouring. By Lemma 3, G itself also contributes $\mathcal{O}(\sqrt{n})$. \square

Lemma 6. $\dots F \dots \mathcal{O}(\log n) \dots$

Consider the leaf partial triangulation G currently being flooded. As already observed, only ancestors of G can be bichromatic. It suffices to prove that every ancestor contributes at most one white region to the current colouring. This is sufficient because there are only $\mathcal{O}(\log n)$ such ancestors and the flooding of G itself contributes only $\mathcal{O}(\log n)$ white regions to the current colouring, by Lemma 3.

Our claim follows from Lemma 2. In particular, the second condition implies that at most one of the white children of a partial triangulation G is not included in the same white region as some white sibling of an ancestor of G . \square

Partition \mathcal{P} can be computed in $\mathcal{O}(n \log n)$ time by repeated application of Lemma 1. In particular, every face of G is contained in exactly one partial triangulation per level of \mathcal{P} , so that the cost of computing every level of \mathcal{P} is $\mathcal{O}(n)$. There are at most $\log_{3/2} n = \mathcal{O}(\log n)$ levels in \mathcal{P} . Once partition \mathcal{P} is given, the remainder of the algorithm is easily carried out in linear time. (Details appear in the full paper.)

We have shown that an $\mathcal{O}(\sqrt{n})$ -bounded $(1, \mathcal{O}(\log n))$ -scattered flooding of a planar triangulation can be computed in $\mathcal{O}(n \log n)$ time. This completes the proof of Theorem 2.

5 Families of Hard Triangulations

As already mentioned in the introduction, it is in general impossible to obtain an $o(\sqrt{n})$ -bounded flooding for a given triangulation. In this section, we prove that the scatter of the flooding in Theorem 2 is also optimal, even if we relax the bound on the boundary size to be $\mathcal{O}(n^{1-\epsilon})$, for any $0 < \epsilon < 1$, and we allow more than one black region, that is, we are only interested in the total number of monochromatic regions. The following theorem states this formally:

Theorem 3. *For any $0 < \epsilon < 1$, there exists a constant c such that for any triangulation T of size $\mathcal{O}(n)$ with boundary size $\mathcal{O}(n^{1-\epsilon})$, the scatter of any flooding of T is at least $c \log n$.*

To prove Theorem 3, we show how to construct, for any pair of parameters n and ϵ , a triangulation T of size $\mathcal{O}(n)$ as in Theorem 3. Then we define a tree X that captures the structure of T ; prove that the scatter of any $\mathcal{O}(n^{1-\epsilon})$ -bounded flooding of T cannot be less than the minimal scatter of a flooding of X , defined below; and finally prove that X does not have an $o(\log n)$ -scattered flooding.

To construct triangulation T , we place triangles in the plane and then triangulate the regions bounded by these triangles. Every such triangle Δ is said to be at a level (i, j) . We compare levels lexicographically; that is, $(i, j) < (i', j')$ if either $i < i'$ or $i = i'$ and $j < j'$.

The first triangle we place is a bounding triangle at level $(0, 0)$. All subsequent triangles are placed inside this triangle. After placing the bounding triangle, we iteratively place two level- $(i + 1, 0)$ -triangles into every level- $(i, 0)$ triangle until the last level $(\ell, 0)$ we produce contains between $n^{\epsilon'}$ and $2n^{\epsilon'}$ triangles, where $\epsilon' = \epsilon/2$. Observe that $\ell \geq \log(n^{\epsilon'}) = \epsilon' \cdot \log n$. We call levels $(0, 0), (1, 0), \dots, (\ell, 0)$

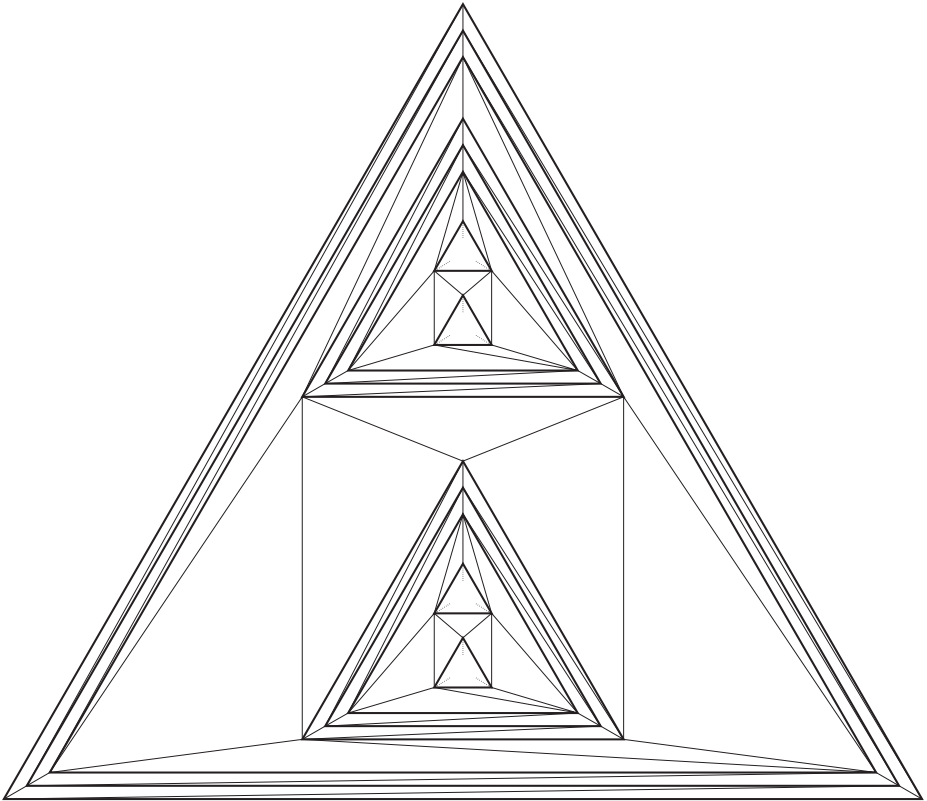


Fig. 1. The construction of a triangulation as in Theorem 3. The nested triangles in the hierarchy are shown in bold. Thin edges show a possible triangulation obtained from this hierarchy of triangles

..... Now we continue by placing triangles at
 $(i, j), j > 0$. Let $m = \lceil n^{1-\epsilon'} \rceil$. Then we place one level- (i, j) triangle Δ , for $0 \leq i \leq \ell$ and $1 < j < m$, into every level- $(i, j - 1)$ triangle Δ' so that the level- $(i + 1, 0)$ triangles contained in Δ' are also contained in Δ . We obtain our final triangulation T by triangulating the regions between triangles at consecutive levels (see Figure 1). To avoid confusion, we refer to the nested triangles we place during the construction of T as and to the triangular faces of T as Our first observation proves that T has the desired number of faces.

Observation 2. T $\mathcal{O}(n)$

..... The number of triangles at level $(\ell, 0)$ is at most $2n^{\epsilon'}$. The number of triangles at all branching levels is at most twice that. Every triangle at a branching level $(i, 0)$ contains $m = \lceil n^{1-\epsilon'} \rceil$ triangles at non-branching levels (i, j) . Hence, the total number of triangles is $\mathcal{O}(n^{\epsilon'} \cdot n^{1-\epsilon'}) = \mathcal{O}(n)$. Since every triangle contributes 3 vertices to the vertex set of T , T has $\mathcal{O}(n)$ vertices and hence, by Euler's formula, $\mathcal{O}(n)$ faces. \square

We call the region bounded by two triangles Δ and Δ' such that Δ' is contained in Δ and these triangles are at levels (i, j) and $(i, j + 1)$, for some i and j , a *tube*. If Δ and Δ' are at levels $(i, 0)$ and $(i, m - 1)$, respectively, we call the region bounded by Δ and Δ' a *boundary tube*. The next observation establishes that every $\mathcal{O}(n^{1-\epsilon})$ -bounded colouring of T has to contain a sufficient number of monochromatic rings, which is the key to lower-bounding the scatter of any $\mathcal{O}(n^{1-\epsilon})$ -bounded flooding of T by the scatter of floodings of certain trees.

Observation 3. *Let c be an $\mathcal{O}(n^{1-\epsilon})$ -bounded colouring of T . Then c defines at least $\omega(n^{1-\epsilon})$ monochromatic rings.*

Assume that there is a tube in T none of whose rings is monochromatic; that is, every ring contains at least one black and at least one white face. Then there is at least one boundary edge between black and white faces in every ring in this tube. Since there are $\lceil n^{1-\epsilon'} - 1 \rceil = \omega(n^{1-\epsilon})$ rings per tube, this contradicts the assumption that c is $\mathcal{O}(n^{1-\epsilon})$ -bounded. □

Next we define a tree X whose floodings lower-bound the scatter of any $\mathcal{O}(n^{1-\epsilon})$ -bounded flooding of T . We begin by constructing a tree X_0 : Tree X_0 contains one node per region that is bounded by a set of triangles and does not contain any triangle. There is an edge between two nodes if the corresponding regions have a common triangle on their boundaries. Tree X is obtained from X_0 by replacing every maximal path whose internal nodes have degree two with an edge. The nodes of X represent the exterior triangle of T , the regions that are bounded by three triangles, and the triangles that do not contain any other triangles. The edges of X represent the tubes of T .

For every $\mathcal{O}(n^{1-\epsilon})$ -bounded colouring c of T , we define a colouring c' of the edges of X as follows: By Observation 3, every tube of T contains either a black or a white ring, or both. We colour the corresponding edge of X black if there is a black ring in the tube and white if all rings in the tube are either white or bichromatic.

A *subtree* of X under colouring c' is a maximal subtree all of whose edges have the same colour. We call a colouring of X $s(n)$ -scattered if it defines at most $s(n)$ monochromatic subtrees of X . A flooding of X is defined analogously to a flooding of T , the only difference being that we colour edges. A flooding is $s(n)$ -scattered if all its colourings are $s(n)$ -scattered.

The next lemma proves that the number of monochromatic subtrees of X defined by c' is a lower bound on the number of monochromatic regions of T defined by colouring c .

Lemma 7. *Let c be an $\mathcal{O}(n^{1-\epsilon})$ -bounded colouring of T and c' the corresponding colouring of X . Let k be the number of monochromatic regions of T defined by c and k' the number of monochromatic subtrees of X defined by c' . Then $k \geq k'$.*

Let k' be the number of monochromatic subtrees of X under colouring c' . We prove that $k \geq k'$. If there are at most two monochromatic subtrees, the lemma is trivial because c cannot define less than one region of T and if X

has two monochromatic subtrees, then T has two regions of different colours. So assume that there are at least three monochromatic subtrees in X . We prove that, for each of these subtrees, there is at least one monochromatic region in T .

Consider two black subtrees X_1 and X_2 and two edges $e_1 \in X_1$ and $e_2 \in X_2$. Since $X_1 \neq X_2$, there has to be at least one white edge e' on the path connecting e_1 and e_2 . Since edges e_1 and e_2 are black, there is at least one black face in each of the tubes represented by these edges. Call these faces f_1 and f_2 . Edge e' is white because there is no black ring in the tube represented by e' . Hence, by Observation 3, this tube contains at least one white ring. Since e' is on the path from e_1 to e_2 in X , this ring separates f_1 from f_2 . Therefore, f_1 and f_2 belong to different black regions of T . This proves that, for every black subtree of X , there is at least one black region in T . A symmetric argument shows that the number of white regions of T is at least the number of white subtrees of X . \square

The following is an easy consequence of Lemma 7.

Corollary 1. T has $\mathcal{O}(n^{1-\epsilon})$ scattered regions in X for any $\epsilon > 0$.

We have to show that X does not have an $o(\log n)$ -scattered flooding. Observe that, by the construction of T and X , X has 2^h nodes, for some integer h .

Lemma 8. X does not admit a $\lfloor h/2 \rfloor$ -scattered flooding.

Observe that, if we root X at the node representing the exterior face, X is a complete binary tree with 2^{h-1} leaves whose root has an extra parent. We call such a tree a *hanger*. Next we use induction on h to prove that a hanger with 2^h nodes does not admit an $\lfloor h/2 \rfloor$ -scattered flooding.

If $h \leq 3$, the claim holds trivially because every colouring is at least 1-scattered and every flooding of a tree with more than one edge is at least 2-scattered. So assume that $h > 3$ and that the claim holds for $h - 2$. Let $F = (c_0, c_1, \dots, c_{2^h-1})$ be a flooding of a hanger H with 2^h nodes. By removing the root of H , its child, and the three edges incident to these vertices, we partition H into two complete subtrees; both subtrees can be partitioned into two hangers with 2^{h-2} nodes. We denote these hangers as H_1, H_2, H_3, H_4 . Since the restriction of F to any H_j is a flooding of H_j with duplicate consecutive colourings, there have to be colourings $c_{i_1}, c_{i_2}, c_{i_3}, c_{i_4}$, $i_1 < i_2 < i_3 < i_4$, such that the restriction of c_{i_j} to H_j defines at least $\lfloor h/2 \rfloor$ monochromatic subtrees of H_j . If, for any colouring c_{i_j} , $H - H_j$ is not monochromatic, c_{i_j} defines at least $\lfloor h/2 \rfloor + 1$ monochromatic subtrees of H . Now we observe that, for colouring c_{i_2} , $H - H_2$ cannot be monochromatic. Indeed, c_{i_2} succeeds c_{i_1} , so H_1 contains at least one edge that is coloured black by c_{i_2} ; c_{i_2} precedes c_{i_3} , so H_3 contains at least one edge that is coloured white by c_{i_2} . Hence, H does not have an $\lfloor h/2 \rfloor$ -scattered flooding. \square

To complete the proof of Theorem 3, it suffices to observe that X has $\Theta(n^{\epsilon'})$ nodes. By Lemma 8, this implies that X does not have an $o(\log n)$ -scattered

flooding and, hence, by Corollary 1, that T does not have an $\mathcal{O}(n^{1-\epsilon})$ -bounded $o(\log n)$ -scattered flooding.

Acknowledgements. We would like to thank two anonymous referees for pointing out the relationship of the triangulation flooding problem to the large number of problems discussed in the introduction.

References

1. P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3d meshes. In *Proceedings of Eurographics*, pages 480–489, 2001.
2. R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics*, 15(2):141–152, 1996.
3. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, 2002.
4. J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.
5. K. Diks, H. N. Djidjev, O. Sykora, and I. Vrto. Edge separators of planar and outerplanar graphs with applications. *Journal of Algorithms*, 14:258–279, 1993.
6. M. Franklin, Z. Galil, and M. Yung. Eavesdropping games: A graph theoretic approach to privacy in distributed systems. *Journal of the ACM*, 47(2):225–243, 2000.
7. C. Gotsman. On the optimality of valence-based connectivity coding. *Computer Graphics Forum*, 22(1):99–102, 2003.
8. F. Hurtado. Open problem posed at the 15th Canadian Conference on Computational Geometry. 2003.
9. D. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492–514, 1999.
10. T. Lengauer. Black-white pebble games and graph separation. *Acta Informatica*, 16(4):465–475, 1981.
11. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.
12. F. Makedon and H. Sudborough. Minimizing width in linear layout. In *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 478–490. Springer-Verlag, 1983.
13. N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, 1988.
14. G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32:265–279, 1986.
15. P. Mutzel. A polyhedral approach to planar augmentation and related problems. In *Proceedings of the 3rd Annual European Symposium on Algorithms*, volume 979 of *Lecture Notes in Computer Science*, pages 497–507, 1995.
16. E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, 2000.

Exact Computation of Polynomial Zeros Expressible by Square Roots

Timo von Oertzen

Saarland University
timovoe@gmx.de

Abstract. In this paper, we give an efficient algorithm to find symbolically correct zeros of a polynomial $f \in R[X]$ which can be represented by square roots. R can be any domain if a factorization algorithm over $R[X]$ is given, including finite rings or fields, integers, rational numbers, and finite algebraic or transcendental extensions of those. Asymptotically, the algorithm needs $O(T_f(d^2))$ operations in R , where $T_f(d)$ are the operations for the factorization algorithm over $R[X]$ for a polynomial of degree d . Thus, the algorithm has polynomial running time for instance for polynomials over finite fields or the rationals.

1 Introduction

The zeros for polynomials of degree two can easily be expressed by square roots. By the formulas of Tartaglia and Ferrari (both published in Cardanos [2] in 1545), also polynomials of degree three and four can be solved by radicals, though these of course include square and cubic roots.

By Galois Theory, N. Abel [1] showed that polynomials of degree five or higher in general can not be solved by radicals. Nevertheless, special instances of such polynomials can be solved. If so, such a representation is usually smaller and easier to handle than the minimal polynomial itself.

If the radicals involved in the roots are already known beforehand, the exact roots can be found by factoring over the respective ring extension. Algorithms for this purpose are for example described in [3].

An algorithm for deciding whether a given polynomial is solvable by radicals and to compute such a solution without knowing the radicals beforehand has been proposed by Landau et al. in [8]. But, since the authors need to approximate the zeros of the polynomial in their approach, it is restricted to polynomial rings over fields where approximative computations are possible; furthermore, very precise approximation of all zeros are needed. Also, this approach was based on early factoring algorithms and therefore had a relatively high running time.

In this paper, we restrict ourselves to the case of square roots. We give an efficient algorithm for testing whether a polynomial of any degree can be solved by square roots and explicitly find these roots if they exist. Our algorithm is based on factorization only, i.e. it works for polynomials over any domain if a factorization algorithm is given, including finite rings or fields, integers, rational

numbers, and algebraic or transcendental extensions of those. For a polynomial of degree d , the algorithm needs asymptotically $O(T_f(d^2))$ operations in the domain, where $T_f(d)$ are the operations for the factorization algorithm in the domain. In the outlook, we will describe how this approach may extend to other radicals as well.

The question of solvability by square roots arises for example in tasks of geometric construction by ruler and compass, tasks that kept mathematicians engaged for centuries. This is due to the fact that with ruler and compass, exactly all square root expressions can be constructed.

Figure 1 depicts an easy example of such construction task: Given a triangle ABC and the intersection of its heights D , is it possible to give a construction (using ruler and compass) to acquire C if A , B and D are given?

Using algebraic means, the question can be easily answered positively. All that has to be done is to embed the construction in a coordinate system, for example by assigning $A = (0, 0)$, $B = (1, 0)$ and $C = (X, Y)$ with two variables X and Y . Then, we can express the coordinates of $D = (h_x(X, Y), h_y(X, Y))$ in terms of X and Y . Assume values a, b for the Position of D are given; than C can be reconstructed from A , B and D exactly when the polynomial equation system

$$h_x(X, Y) = a$$

$$h_y(X, Y) = b$$

can be solved by square roots. By means of elimination theory, we can reduce this task to a question whether the root of a single polynomial can be represented by square roots. For a more thorough description of the interrelation of solving polynomials with square roots and the construction problem, see [11].

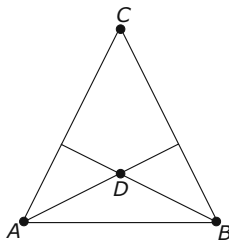


Fig. 1. The intersection of heights in a triangle Given the points A , B and D , then C can be constructed be ruler and compass such that D is the intersection of heights in the triangle ABC

In the following, we will first give a description of the \dots over a field \mathbb{K} , i.e. the closure of \mathbb{K} by square roots. We will talk about representing elements in such fields and define a property called \dots shared by most, though not all, elements of the Euclidian field. Complete elements provide some

further properties; for example, the representation of complete elements is unique in a certain sense.

In the third section, we will give a quick test to verify whether a given polynomial has a root representable by square roots without explicitly computing such a representation. To the end of constructing such a representation, we give some theorems that enable us to compute the representation of a zero for a given polynomial by square roots.

In the fourth section, we will give the algorithm in compact form and prove time bounds of it. We give some possible improvements of the algorithm for special situations.

Finally, we will give a discussion and outlook on possible further investigations in the field.

2 Theoretical Aspects of the Euclidian Field

In this section, a short overview is given on aspects of the Euclidian Field of a field \mathbb{K} , defined by

Definition 21. $\mathbb{K} \subseteq \mathbb{E}_{\mathbb{K}} \subseteq \overline{\mathbb{K}}$

$$\mathbb{E}_{\mathbb{K}} = \bigcap \{ \mathbf{K} \mid \mathbf{K} \supseteq \mathbb{K}, \forall a \in \mathbf{K} \exists b \in \mathbf{K} : b^2 = a \}$$

Euclidian Field \mathbb{K}

$$\xi \in \mathbb{E}_{\mathbb{K}} \implies \deg(\xi) \in \mathbb{K}$$

2.1 The Representation Theorem

Elements of the Euclidian Field can be represented as follows:

Theorem 22 (Representation Theorem). $\mathbf{R} \subseteq \mathbb{E}_{\mathbb{K}} \subseteq \mathbb{K}$
 $\mathbf{R} = \{ \xi \in \mathbb{E}_{\mathbb{K}} \mid \deg(\xi) \leq 2^d \}$

$$\xi = \frac{1}{n} \left(k_0 + \sum_{i=1}^d k_i \sqrt{w_i} \right)$$

$n, k_0 \in \mathbf{R}, w_i, k_i \in \mathbb{E}_{\mathbb{K}}, \deg(w_i) \leq 2^{i-1}, \deg(k_i) \leq 2^{i-1}$
 $i = 1, \dots, d$

$$w_i = k_j^2 \cdot w_j \quad \text{for } j < i$$

$\mathbf{R}[\sqrt{w_1}, \dots, \sqrt{w_{i-1}}]$

For a proof, see [11].

Lemma 23. $\xi \in \mathbb{E}_{\mathbb{K}}, \deg(\xi) \leq 2^d, \xi = \xi_1 + \sqrt{\xi_2}$
 $\deg(\xi_2) = 2^{d-1}$

(.....) Take any representation of ξ and order the square roots by degree. If $\deg(\xi_2) < 2^{d-1}$, there must be two successive roots $k_i\sqrt{w_i} + k_{i+1}\sqrt{w_{i+1}}$ with equal degree $\deg(w_i) = \deg(w_{i+1})$ in the representation of ξ . Replace this term by

$$\sqrt{(k_i\sqrt{w_i} + k_{i+1}\sqrt{w_{i+1}})^2} = \sqrt{k_i^2w_i + k_{i+1}^2w_{i+1} + 2k_ik_{i+1}\sqrt{w_iw_{i+1}}}$$

As can be checked, the resulting term still is a representation as in the representation theorem. Continuing in this way leads to a representation as desired. \square

Furthermore, it can be shown that if ξ_1 and ξ_2 are to be represented in the same way, this representation is unique.

For each square root in the representation, we can choose a sign separating the two possible branches of the square root. For ease of notation, choose any fixed square root function for the $\sqrt{\cdot}$ symbol; then, exchanging the signs for the square roots in the representation theorem enables us to switch between the algebraic conjugates of whole ξ . By this, we get hold of the minimal polynomial of ξ :

Lemma 24. $\xi \in \mathbb{E}_{\mathbb{K}}$, $\xi = \frac{1}{n} \left(k_0 + \sum_{i=1}^d k_i\sqrt{w_i} \right)$
 $(v_1, \dots, v_d) \in \{0, 1\}^d$

$$\xi^{(v_1, \dots, v_d)} := k_0 + \sum_{i=1}^d (-1)^{v_i} k_i^{(v_1, \dots, v_{i-1})} \sqrt{w_i^{(v_1, \dots, v_{i-1})}}$$

..... $\xi \sim \eta$, $\eta = \xi^{(v_1, \dots, v_d)}$ v_i

$$f = \prod_{\eta \sim \xi} (X - \eta)$$

..... ξ $\xi^{(v_1, \dots, v_d)}$ conjugates ξ

(.....) The element ξ is a zero of f , and f is monic and of degree 2^d , which is the degree of the field extension $[\mathbb{K} : \mathbb{K}(\xi)]$. Therefore, it suffices to show that $f \in \mathbb{K}[X]$. This can be checked by an easy computation, since for all $\eta \sim \xi$, the $X - \eta$ carry all possible signs in front of the square roots appearing in the term and are therefore successively eliminated when computing the product. \square

2.2 Completeness

In most cases, the representation given in the representation theorem is unique up to quadratic factors in the square roots. Though, this does not hold if for one of the expressions $\sqrt{w_d}$ in the representation theorem, not all square roots of conjugates of w_d are linearly independent. If this situation occurs, we call the element..... Completeness can be equivalently described in following ways:

Lemma 25. . $\xi \in \mathbb{E}_{\mathbb{K}}$, $\xi = k_0 + \sum_{i=1}^d k_i \sqrt{w_i}$.

$$\mathbb{K}_i = \mathbb{K}(\sqrt{w'_j} \mid j \leq i, w'_j \sim w_j)$$

$$K_d \dots \xi$$

$$[\mathbb{K}_{i+1} : \mathbb{K}_i] = 2^{2^i} \forall i$$

$$[\mathbb{K}_d : \mathbb{K}] = 2^{2^d - 1}$$

$$\xi \dots 2 \dots S_{2^d}$$

ξ complete (\dots)
 $2 \dots$)

K_d contains all zeros of the minimal polynomial of ξ (i.e. all conjugates), therefore it is a superfield of the splitting field. On the other hand, by eliminating the square roots by appropriate adding of the conjugates, all conjugates of all $\sqrt{w_d}$ can be represented by the conjugates of ξ , thus K_d is a subfield of the splitting field. Hence, K_d is the splitting field.

The equivalence of the three characterizations of completeness can be checked as follows. If (1), then (2) follows since

$$[\mathbb{K}_d : \mathbb{K}] = \prod_{i=0}^{d-1} 2^{2^i} = 2^{\sum_{i=0}^{d-1} 2^i} = 2^{2^d - 1}$$

(2) \Rightarrow (3) holds since the Galois group of ξ must have $2^{2^d - 1}$ elements, and the factor 2 is contained exactly $2^d - 1$ times in $2^d!$. If (3) holds, (1) follows, since the Galois group can only be of maximal size if each field extension step is of maximal size. □

If an element is complete, we get some additional information about it:

Lemma 26. . $\xi \in \mathbb{E}_{\mathbb{K}}$, $\xi = k_0 + \sum_{i=1}^d k_i \sqrt{w_i}$.

$$1 \leq j \leq d, w_j \dots 2^{j-1}$$

$$1 \leq j \leq d, k_0 + \sum_{i=1}^j k_i \sqrt{w_i} \dots k_j \neq 0$$

If ξ is complete, then by Lemma 25 \mathbb{K}_d is of dimension $2^{2^d - 1}$ as a \mathbb{K} -vector space. Thus, the $2^{2^d - 1}$ elements $\sqrt{w'_j}$ for $1 \leq j \leq d$ and $w'_j \sim w_j$ form a basis of \mathbb{K}_d , and the representation of ξ by the representation theorem (which is a representation by this basis) is unique.

(2) follows immediately by the definition of completeness. (3) follows from (2) if $k_j \neq 0$, since if $\sqrt{w_j}$ is complete, so is each sum of it and terms containing only square roots $\sqrt{w_i}$ with $i < j$. □

3 Towards the Solving Algorithm

In the following section, a quick testing algorithm for polynomials over \mathbb{Q} or finite extensions of \mathbb{Q} will be introduced to check whether the polynomial has roots representable by square roots.

Furthermore, for a polynomial that actually has zeros representable in the Euclidian Field over the ground field, a theorem will be given to isolate the last square root in a representation of such a zero, in this way reducing the problem of finding the representation to a problem of reduced polynomial degree.

First, we assume a given polynomial to be irreducible by factoring it and finding the zeros of the factors independently. If an irreducible polynomial has a degree which is not a power of two, it can certainly be dismissed, since by lemma 24 the degree of the minimal polynomial of a zero representable by square roots is a power of two.

3.1 Quick Testing Over Extensions of the Rationals

For a polynomial over the rationals or a finite extension of the rationals, we can extend the test of degree to the factors of the polynomial over finite fields. Since if a polynomial over \mathbb{Q} has zeros in $\mathbb{E}_{\mathbb{Q}}$, so has its image in every finite field. But although we assume the polynomial to be irreducible over \mathbb{Q} , of course it may split over a finite field. If any factor over a finite field has a degree which is not a power of two, we can conclude that it has no zeros representable by square roots, and thus neither the original polynomial over \mathbb{Q} .

We use this to provide a quick pretest as follows: Take an irreducible polynomial over \mathbb{Q} (or an appropriate finite extension of \mathbb{Q}) and factor it over several finite fields, say over $\mathbb{Z}/p\mathbb{Z}$ for the first hundred primes. If any of these factors has a degree which is not a power of 2, dismiss the polynomial. Otherwise, the polynomial is very likely to actually have a zero representable by square roots, although it is not certain. In the literature, this test is often used to learn something about the structure of the Galois group of a polynomial (see [9, 10]); we can make use of it here since the polynomial will have a root expressible by square roots exactly if its Galois group is a 2-subgroup.

We formalize the algorithm in the following lemma:

Lemma 31. $\mathbb{K} = \mathbb{Q}(Y_1, \dots, Y_n) \xrightarrow{\varphi} \mathbb{K}_p = \mathbb{F}_p(Y_1, \dots, Y_n)$
 $\mathbb{Z}(Y_1, \dots, Y_n) \rightarrow \mathbb{F}_p(Y_1, \dots, Y_n) \xrightarrow{f} \mathbb{K}[X]$
 $\mathbb{Z} \xrightarrow{\xi \in \mathbb{E}_{\mathbb{K}}} \mathbb{K}_p$
 $\varphi(f) \in \mathbb{K}_p$

The proof of this lemma can be found in [11].

If a polynomial has roots representable by square roots, it will pass the test by the above lemma. If, on the other hand, the polynomial has no such roots, the test may fail. Yet, by Tchebotarevs density theorem (see [9, 10]), we know that by factoring over a sufficiently general prime, the probability to get a factor of degree not divisible by two is equal to the ratio of elements in the Galois group

which have no cycle divisible by two. For generic polynomials of degree four (i.e. polynomials with S_4 as their Galois group), this ratio is $\frac{1}{3}$ (8 of the 24 elements in the S_4 have a cycle of length three), i.e. the probability that the test fails on 25 primes is about $4 \cdot 10^{-3}$ %. For degree 8, about $\frac{3}{4}$ of the elements of the S_8 have a cycle of length 3, 5, 6 or 7, so the probability that the test fails on 25 primes is about $9 \cdot 10^{-14}$ %. For higher degrees, the test is even more successful. Experiments of the test on randomly chosen polynomials (see [11]) reveal the same results.

3.2 Constructive Algorithm for Solving Polynomials by Square Roots

The following theorem is the core of the algorithm to find the zeros of a polynomial explicitly. It shows how to find the minimal polynomial of the last square root in the representation, in this way reducing the problem to a polynomial of half degree.

Theorem 32. . . . $\xi \in \mathbb{E}_{\mathbb{K}}$. . . $\xi = \xi_1 + \sqrt{\xi_2}$. . .

$$f_{4\xi_2} := \prod_{(v_1, \dots, v_{d-1}) \in \{0,1\}^{d-1}} X - \left(\xi^{(v_1, \dots, v_{d-1}, 0)} - \xi^{(v_1, \dots, v_{d-1}, 1)} \right)^2$$

$$\dots \dots \dots 4\xi_2$$

As can be checked by computation (see [11]), we have

$$f_{4\xi_2} = \prod_{(v_1, \dots, v_{d-1}) \in \{0,1\}^{d-1}} X - \left(\xi^{(v_1, \dots, v_{d-1}, 0)} - \xi^{(v_1, \dots, v_{d-1}, 1)} \right)^2$$

$$= \prod_{(v_1, \dots, v_{d-1}) \in \{0,1\}^{d-1}} X - 4\xi_2^{\xi^{(v_1, \dots, v_{d-1})}}$$

Which is the product of all conjugates of $4\xi_2$, thus a power of the minimal polynomial; since Lemma 23 states that $\deg(\xi_2) = 2^{d-1}$, it is exactly the minimal polynomial. □

The complete algorithm is outlined as follows: Given an irreducible polynomial f , assume it is the minimal polynomial of an element $\xi \in \mathbb{E}_{\mathbb{K}}$. We then compute $f_{4\xi_2}$ by the way described below. It can be easily proved (see [11]) that for every $\xi \in \mathbb{E}_{\mathbb{K}}$, there is one representation of ξ like in the representation theorem such that every square root appearing in ξ also appears in ξ_2 .

Since $\deg(f_{4\xi_2}) = \frac{1}{2} \deg(f_\xi)$, we can solve $f_{4\xi_2}$ recursively and thereby find all square roots in ξ (namely, those appearing in ξ_2 and $\sqrt{\xi_2}$ itself).

Then, we can find the zero of f by factoring f over the extension field given by these square roots. A description for algorithms to factor over number field extensions can for example be found in [3].

What's left is to compute $f_{4\xi_2}$ from ξ 's minimal polynomial f . To this end, we introduce

$$f_- = \prod_{\eta_1, \eta_2 \sim \xi} X - (\eta_1 - \eta_2)$$

Note that since f_- contains all differences of conjugates of ξ as zeros, it is a multiple of

$$\begin{aligned}
 f_{2\sqrt{\xi_2}} &:= \prod_{(v_1, \dots, v_{d-1}) \in \{0,1\}^{d-1}} \left(X - \left(\xi^{(v_1, \dots, v_{d-1}, 0)} - \xi^{(v_1, \dots, v_{d-1}, 1)} \right) \right) \\
 &\quad \cdot \left(X + \left(\xi^{(v_1, \dots, v_{d-1}, 0)} - \xi^{(v_1, \dots, v_{d-1}, 1)} \right) \right) \\
 &= \prod_{(v_1, \dots, v_{d-1}) \in \{0,1\}^{d-1}} X^2 - \left(\xi^{(v_1, \dots, v_{d-1}, 0)} - \xi^{(v_1, \dots, v_{d-1}, 1)} \right)^2
 \end{aligned}$$

which is $f_{4\xi_2}$ up to replacing X^2 with X . So, we can achieve $f_{4\xi_2}$ by factoring f_- .

To find f_- from the coefficients of f , we use a trick by defining

Definition 33. $\eta_0, \dots, \eta_{m-1}$ are the coefficients of f_- .

$$g(k, n) := \sum_{i_0 \neq \dots \neq i_k} \eta_{i_0} \cdots \eta_{i_k}^n$$

$i_0 \neq \dots \neq i_k$

Notice that up to a factor $\frac{1}{k!}$, the $g(k, 1)$ are the elementary symmetric polynomials in η_i and thus the coefficients of f_- , which we are interested in.

The $g(k, n)$ can be computed from the $g(1, n)$ by the following recursion:

Lemma 34. $g(k, n)$

$$\begin{aligned}
 g(k, n) &= (-1)^{k-1} (k-1)! \cdot g(1, k+n-1) \\
 &\quad + \sum_{i=0}^{k-2} (-1)^i \frac{(k-1)!}{(k-1-i)!} g(k-1-i, 1) \cdot g(1, n+i)
 \end{aligned}$$

The proof is done by induction on the first argument k of g and can be found in [11].

The $g(f, 1)$ on the other hand can be easily computed from the coefficients of the original polynomial f . To this end, we first recall which compute the power sums of a polynomial by its coefficients:

Lemma 35 (Newton's Equations). f ξ_i, σ_k ξ_i f s_k ξ_i

$$s_k = \sum_{i=1}^d \xi_i^k$$

$$s_k = \begin{cases} (-1)^{k-1} k \sigma_k - \sum_{i=1}^{k-1} (-1)^i \sigma_i s_{k-i} & 1 \leq k \leq d \\ (-1)^{d-1} \sigma_d s_{k-d} - \sum_{i=1}^{d-1} (-1)^i \sigma_i s_{k-i} & k > d \end{cases}$$

A proof of this lemma can for example be found in [4].

With the power sums, we can compute $g(1, n)$ from the coefficients of f :

Lemma 36. ξ_0, \dots, ξ_{n-1} $f = f - \dots$
 s_k f

$$g(1, n) = ds_n (1 + (-1)^n) + \sum_{k=1}^{n-1} (-1)^k \binom{n}{k} s_{n-k} s_k$$

$$g(1, n) = 0 \quad n$$

We combine these results to get a recursion for the coefficients of f_- , which are $\frac{1}{k!}g(k, 1)$. We denote the coefficients by $\gamma(k)$. By above computation, we know the $g(1, n)$; then, the coefficients of f_- can be computed as follows:

Lemma 37. $\gamma(k) = \frac{1}{k!} g(k, 1)$

$$\gamma(k) = -\frac{1}{k} \left(g(1, k) + \sum_{\substack{i=0 \\ i \text{ odd}}}^{k-2} \gamma(k-1-i)g(1, 1+i) \right)$$

$$\gamma(k) = 0 \quad k$$

For the proofs of this last two lemmata, we again refer to [11].

4 The Algorithm to Compute the Zeros Representable by Square Roots

Algorithm to Compute All Zeros representable by Square Roots

Input: Polynomial $f \in \mathbb{K}[X]$

1. Factor f . Return all zeros of linear factors and proceed with every irreducible factor of f with its degree being a power of 2.
2. Normalize f to $f = X^d + \sum_{i=1}^d \sigma_i X^{d-i}$. Let $d_2 = \frac{d(d-1)}{2}$.
3. Compute for $1 \leq k \leq 2d_2$

$$s_k = \begin{cases} (-1)^{k-1} k \sigma_k - \sum_{i=1}^{k-1} (-1)^i \sigma_i s_{k-i} & 1 \leq k \leq d \\ (-1)^{d-1} \sigma_d s_{k-d} - \sum_{i=1}^{d-1} (-1)^i \sigma_i s_{k-i} & k > d \end{cases}$$

4. Compute for $1 \leq n \leq d_2$

$$g(1, 2n - 1) = 0$$

$$g(1, 2n) = 2ds_{2n} + \sum_{k=1}^{2n-1} (-1)^k \binom{2n}{k} s_{2n-k} s_k$$

5. Compute \widetilde{f}_- for $1 \leq n \leq d_2$

$$\gamma(2n - 1) = 0$$

$$\gamma(2n) = -\frac{1}{2n} \left(g(1, 2n) + \sum_{i=0}^{2n-2} \gamma(2n - 1 - i)g(1, 1 + i) \right)$$

6. Compute $\widetilde{f}_- = X^{d_2} + \sum_{i=1}^{d_2} \gamma_-(2i)X^{d_2-i}$ (Notice that this is the nontrivial factor of $f_- (\sqrt{X})$)

7. Factor \widetilde{f}_- ; if there is no factor with degree $\frac{d}{2}$, return unsolvable. Else, choose one such factor and solve recursively to a solution ξ .

8. Factor f in the number field given by the square roots in the normal form of ξ and $\sqrt{\xi}$ itself. If there is no linear factor, return unsolvable. Else, return the zero of one linear factor.

We will give a short example on how the algorithm works. Consider the irreducible polynomial $f = X^4 - 4X^3 + 8X + 2$. For the sake of completeness, we give here the intermediate values in the algorithm $\sigma_n, s_n, g(1, n)$, and γ_n :

n	1	2	3	4	5	6	7	8	9	10	11	12
σ_n	-4	0	8	2	0	0	0	0	0	0	0	0
s_n	-4	16	-40	120	-344	1024	-3056	9232	-28048	85696	-262816	808416
$g(1, n)$	0	96	0	1216	0	17280	0	262656	0	4187136	0	69025792
γ_n	0	-48	0	848	0	-6720	0	22592	0	-23040	0	7168

Thus, $\widetilde{f}_- = X^6 - 48X^5 + 848X^4 - 6720X^3 + 22592X^2 - 23040X + 7168$, which factors to $\widetilde{f}_- = (X^2 - 24X + 112) (X^2 - 12X + 8)^2$. The first factor has the zeros $4(3 \pm \sqrt{2})$; if we factor f over $\mathbb{Q}(\sqrt{2}, \sqrt{3 + \sqrt{2}})$, we find that

$$f = (X^2 - 2X - 2 + \sqrt{2})(X - 1 + \sqrt{3 + \sqrt{2}})(X - 1 - \sqrt{3 + \sqrt{2}})$$

and thus all zeros of f are $1 \pm \sqrt{3 \pm \sqrt{2}}$.

Theorem 41.

Let $f \in \mathbb{K}[X]$ be an irreducible polynomial of degree d over a number field \mathbb{K} . Let $T_f(d)$ be the set of all d -tuples $(\theta_1, \dots, \theta_d) \in \mathbb{K}^d$ such that θ_i are the roots of f in \mathbb{K} . Let $O(T_f(d^2))$ be the set of all d^2 -tuples $(\theta_1, \dots, \theta_{d^2}) \in \mathbb{K}^{d^2}$ such that θ_i are the roots of f in \mathbb{K} .

We prove by induction on the degree; for degree one, the algorithm is clearly correct.

Assume that an irreducible polynomial f of higher degree has a zero $\xi = \eta_1 \pm \sqrt{\eta_2}$ representable by square roots. Then, all zeros of f are representable

by square roots (for these are the conjugates of ξ), and thus also all zeros of all factors of f_- . By Theorem 32, the minimal polynomial of $4\sqrt{\eta_2}$ is among these factors, so there is at least one factor of $\widetilde{f_-}$ with degree $\frac{d}{2}$. By the induction hypothesis, we find η_2 on this factor recursively, and hence the field extension of all successive square roots in the representation of ξ . Thus, the final factorization reveals a linear factor containing ξ . Of course, if f had no zeros representable by square roots, this factorization reveals no linear factor.

The loops to compute f_- take $O(d^2)$ operations, which is less than $T_f(d^2)$ if $T_f(d) \in \theta(d)$. We must factor the initial polynomial, which takes $O(d)$ operations, and f_- , which takes $O\left(T_f\left(\frac{d(d-1)}{2}\right)\right)$ operations. Finally, we must factor f over the extension of $\log(d)$ square roots. By factoring algorithms as described for example in [3], this can be done in $T_f(2^{\log(d)}d) = T_f(d^2)$ operations. We have only one recursion with half the problem size, so the running time of the whole process is bound by twice the running time of the first step of recursion. \square

Remark 42.
$$f = \left(\frac{f_-}{f} \right)^{\frac{d}{2}}$$

Remark 43.
$$f = \xi = \eta_1 \pm \sqrt{\eta_2} = \frac{f_-}{f_+} = \frac{f_+}{f_-} = \frac{p_+}{p_-} = \frac{2\eta_1}{p_+ - p_-}$$

$$O(d^{\log d}) \left(\frac{d}{2} \right)$$

Remark 44.
$$\left(\frac{f_-}{f} \right) \in \mathbb{Q} \left(\frac{f_-}{f} \right)$$

$$f_- \in \mathbb{Q} \left(\frac{f_-}{f} \right) \left(\frac{f_-}{f} \right)^{\frac{d}{2}}$$

$$\mathbb{Q} \left(\frac{f_-}{f} \right)$$

5 Summary and Outlook

We gave an algorithm to symbolically find all zeros of a polynomial $f \in R[X]$ representable by square roots. Here, $R[X]$ can be any polynomial ring where

a factorization algorithm exists. The algorithm is asymptotically as fast as the factorization of a polynomial of degree d^2 , if d is the degree of f . We also introduced a quick test for $\mathbb{Z}[X]$ resp. $\mathbb{Q}[X]$ to quickly identify whether f has zeros representable by square roots.

The algorithm can be extended to find zeros representable by any radicals. Instead of combining two zeros, we must combine as many zeros as necessary to eliminate the outermost root. Consider for example $\xi = \sqrt{2} + \sqrt[3]{1 + \sqrt{2}}$; if ζ is one primitive third root of unity, all conjugates of ξ are:

$$\begin{aligned} \xi_1 &= \sqrt{2} + \sqrt[3]{1 + \sqrt{2}} & \xi_4 &= -\sqrt{2} + \sqrt[3]{1 - \sqrt{2}} \\ \xi_2 &= \sqrt{2} + \zeta \sqrt[3]{1 + \sqrt{2}} & \xi_5 &= -\sqrt{2} + \zeta \sqrt[3]{1 - \sqrt{2}} \\ \xi_3 &= \sqrt{2} + \zeta^2 \sqrt[3]{1 + \sqrt{2}} & \xi_6 &= -\sqrt{2} + \zeta^2 \sqrt[3]{1 - \sqrt{2}} \end{aligned}$$

Here, we can eliminate the third root by adding $\xi_1 + \xi_2 + \xi_3 = 3\sqrt{2}$. Isolation of the third root is then as simple.

Radicals themselves are of course only special cases of easily representable roots, namely roots of polynomials of the form $X^d - a$ for $\sqrt[d]{a}$. The algorithm might also extend to other simple roots; lets for example introduce a symbol $\sqrt[5]{a}, b, c$ that represents one root of the polynomial $X^5 + aX^2 + bX + c$. It would be interesting to investigate possibilities to find roots representable by radicals and this new symbol.

Note that a representation with roots is possibly much shorter than a representation by the minimal polynomial. For example, the term $2 + \sqrt{2} + \sqrt{3}\sqrt{\sqrt{5} + \sqrt{7}}$ is reasonably short to write down, and it is relatively easy to work on it symbolically. Nevertheless, its minimal polynomial is a dense polynomial of degree 32, its highest coefficient consisting of more than 14 digits. Thus, it seems worthwhile to consider a symbolic representation of numbers, if possible, by radicals or extensions as mentioned above.

Acknowledgements

The author wants to thank Günter Hotz, Frank-Olaf Schreyer, Elmar Schömer, Stavros Papadakis, Tobias Gärtner and Manuel Bodirsky for their great help on this paper.

References

1. N.H. Abel: Beweis der Unmöglichkeit, algebraische Gleichungen von höheren Graden als dem vierten allgemein aufzulösen. Journal der reinen und angewandten Mathematik 1, 1826.
2. G. Cardano: Ars Magna, 1545.

3. H.Cohen: A Course in Computational Algebraic Number Theory. Springer, 2000.
4. D. Cox, J. Little, D. O'Shea: Ideals, Varieties, and Algorithms. Springer-Verlag, 1997.
5. D. Cox, J. Little, D. O'Shea: Using Algebraic Geometry. Springer-Verlag, 1998.
6. J. v. z. Gathen, J. Gerhard: Modern Computer Algebra. Cambridge University Press, 1999.
7. K.O. Geddes, S.R. Czapor, G. Labahn: Algorithms for Computer Algebra. Kluwer, 1992.
8. S. Landau, G. Miller: Solvability by Radicals is in Polynomial Time. Journal of Computer and Systems Sciences, vol. 30, No. 2 (1985), s. 179-208.
9. S. Lang: Algebraic Number Theory. Springer Verlag, New York, 1986.
10. J. Neukirch: Algebraische Zahlentheorie, Springer Verlag, Berlin, 1992
11. T. v. Oertzen: Das Konstruktionsproblem. Phd Thesis at the Saarland University, 2003.
12. H.J. Reifen, G. Scheja, U. Vetter: Algebra. Verlag Bibliographisches Institut AG, 1984.

Many-to-Many Disjoint Path Covers in a Graph with Faulty Elements^{*}

Jung-Heum Park¹, Hee-Chul Kim², and Hyeong-Seok Lim³

¹ The Catholic University of Korea, Korea
j.h.park@catholic.ac.kr

² Hankuk University of Foreign Studies, Korea
hckim@hufs.ac.kr

³ Chonnam National University, Korea
hslim@chonnam.ac.kr

Abstract. In a graph G , k vertex disjoint paths joining k distinct source-sink pairs that cover all the vertices in the graph are called a *many-to-many k -disjoint path cover (k -DPC)* of G . We consider an f -fault k -DPC problem that is concerned with finding many-to-many k -DPC in the presence of f or less faulty vertices and/or edges. We consider the graph obtained by merging two graphs H_0 and H_1 , $|V(H_0)| = |V(H_1)| = n$, with n pairwise nonadjacent edges joining vertices in H_0 and vertices in H_1 . We present sufficient conditions for such a graph to have an f -fault k -DPC and give the construction schemes. Applying our main result to interconnection graphs, we observe that when there are f or less faulty elements, all of recursive circulant $G(2^m, 4)$, twisted cube TQ_m , and crossed cube CQ_m of degree m have f -fault k -DPC for any $k \geq 1$ and $f \geq 0$ such that $f + 2k \leq m - 1$.

1 Introduction

One of the central issues in various interconnection networks is finding node-disjoint paths concerned with the routing among nodes and the embedding of linear arrays. Node-disjoint paths can be used as parallel paths for an efficient data routing among nodes. Also, each path in node-disjoint paths can be utilized in its own pipeline computation. An interconnection network is often modeled as a graph, in which vertices and edges correspond to nodes and links, respectively. In the rest of this paper, we will use standard terminology in graphs (see [1]).

Disjoint paths can be categorized as three types: one-to-one, one-to-many, and many-to-many. One-to-one type deals with the disjoint paths joining a single source s and a single sink t . One-to-many type considers the disjoint paths joining a single source s and k distinct sinks t_1, t_2, \dots, t_k . Most of the works done on disjoint paths deal with the one-to-one or one-to-many. For a variety of networks one-to-one and one-to-many disjoint paths were constructed, e.g., hypercubes [3],

^{*} This work was supported by grant No. R01-2003-000-11676-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

star networks [2], etc. Many-to-many type deals with the disjoint paths joining k distinct sources s_1, s_2, \dots, s_k and k distinct sinks t_1, t_2, \dots, t_k . In many-to-many type, several problems can be defined depending on whether specific sources should be joined to specific sinks or a source can be freely matched to a sink. The works on many-to-many type have a relative paucity because of its difficulty and some results can be found in [4, 7].

All of three types of disjoint paths in a graph G can be accommodated with the covering of vertices in G . A disjoint path cover problem in a graph G is to find disjoint paths containing all the vertices in G . A disjoint path cover problem originated from an interconnection network is concerned with the application where the full utilization of nodes is important. For an embedding of linear arrays in a network, the cover implies every node can be participated in a pipeline computation. One-to-one disjoint path covers in recursive circulants[8, 12] and one-to-many disjoint path covers in some hypercube-like interconnection networks[9] were studied.

Given a set of k sources $S = \{s_1, s_2, \dots, s_k\}$ and a set of k sinks $T = \{t_1, t_2, \dots, t_k\}$ in a graph G such that $S \cap T = \emptyset$, we are concerned with many-to-many disjoint paths P_1, P_2, \dots, P_k in G , P_i joining s_i and t_i , $1 \leq i \leq k$, that cover all the vertices in the graph, that is, $\bigcup_{1 \leq i \leq k} V(P_i) = V(G)$ and $V(P_i) \cap V(P_j) = \emptyset$ for all $i \neq j$. Here $V(P_i)$ and $V(G)$ denote the vertex sets of P_i and G , respectively. We call such k disjoint paths a k -disjoint path cover (in short, k -DPC) of G .

On the other hand, embedding of linear arrays and rings into a faulty interconnection network is one of the important problems in parallel processing [5, 6, 11]. The problem is modeled as finding as long fault-free paths and cycles as possible in the graph with some faulty vertices and/or edges. A graph G is called f -fault hamiltonian (resp. f -fault hamiltonian-connected) if there exists a hamiltonian cycle (resp. if each pair of vertices are joined by a hamiltonian path) in $G \setminus F$ for any set F of faulty elements such that $|F| \leq f$. For a graph G to be f -fault hamiltonian (resp. f -fault hamiltonian-connected), it is necessary that $f \leq \delta(G) - 2$ (resp. $f \leq \delta(G) - 3$), where $\delta(G)$ is the minimum degree of G .

To a graph G with a set of faulty elements F , the definition of a many-to-many disjoint path cover can be extended. Given a set of k sources $S = \{s_1, s_2, \dots, s_k\}$ and a set of k sinks $T = \{t_1, t_2, \dots, t_k\}$ in $G \setminus F$ such that $S \cap T = \emptyset$, a many-to-many k -disjoint path cover joining S and T is k disjoint paths P_i joining s_i and t_i , $1 \leq i \leq k$, such that $\bigcup_{1 \leq i \leq k} V(P_i) = V(G) \setminus F$, $V(P_i) \cap V(P_j) = \emptyset$ for all $i \neq j$, and every edge on each path P_i is fault-free. Such a many-to-many k -DPC is denoted by k -DPC $[\{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\} | G, F]$. A graph G is called f - k -DPC if for any set F of faulty elements such that $|F| \leq f$, G has k -DPC $[\{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\} | G, F]$ for every k distinct sources s_1, s_2, \dots, s_k and k distinct sinks t_1, t_2, \dots, t_k in $G \setminus F$.

Proposition 1. Let G be a graph with minimum degree $\delta(G) \geq f + 2k \leq \delta(G) + 1$ and k distinct sources s_1, s_2, \dots, s_k and k distinct sinks t_1, t_2, \dots, t_k in $G \setminus F$.

Proposition 2. Let G be a graph with minimum degree $\delta(G) \geq f + 1$ and k distinct sources s_1, s_2, \dots, s_k and k distinct sinks t_1, t_2, \dots, t_k in $G \setminus F$.

$$() G \dots f \dots k(\geq 2) \dots G \dots f \dots k-1$$

Proposition 3. $G \dots f \dots k(\geq 2)$
 $s \dots t$
 $k-1 \dots ((x_1, y_1), (x_2, y_2), \dots, (x_{k-1}, y_{k-1}))$
 $G \setminus F \dots s \dots t$
 $F \dots |F| \leq f$
 $(s, \dots, x_1, y_1, \dots, x_{k-1}, y_{k-1}, \dots, t)$

We are given two graphs G_0 and G_1 with n vertices. We denote by V_i and E_i the vertex set and edge set of G_i , $i = 0, 1$, respectively. We let $V_0 = \{v_1, v_2, \dots, v_n\}$ and $V_1 = \{w_1, w_2, \dots, w_n\}$. With respect to a permutation $M = (i_1, i_2, \dots, i_n)$ of $\{1, 2, \dots, n\}$, we can “merge” the two graphs into a graph $G_0 \oplus_M G_1$ with $2n$ vertices in such a way that the vertex set $V = V_0 \cup V_1$ and the edge set $E = E_0 \cup E_1 \cup E_2$, where $E_2 = \{(v_j, w_{i_j}) | 1 \leq j \leq n\}$. We denote by $G_0 \oplus G_1$ a graph obtained by merging G_0 and G_1 w.r.t. an arbitrary permutation M . Here, G_0 and G_1 are called \dots of $G_0 \oplus G_1$.

In this paper, we will show that by using f' -fault many-to-many k' -DPC of G_i for all f' and k' such that $f' + 2k' \leq f + 2k$, and fault-hamiltonicity of G_i , we can always construct an $f + 1$ -fault many-to-many k -DPC in $G_0 \oplus G_1$ and an f -fault many-to-many $k + 1$ -DPC in $H_0 \oplus H_1$, where $H_0 = G_0 \oplus G_1$ and $H_1 = G_2 \oplus G_3$. Precisely speaking, we will prove the following two theorems. Note that $\delta(G_0 \oplus G_1) = \delta + 1$ and $\delta(H_0 \oplus H_1) = \delta + 2$, where $\delta = \min_i \delta(G_i)$.

Theorem 1. $k \geq 2 \dots f \geq 0, \dots k = 1 \dots f \geq 2, \dots G_i \dots n \dots i = 0, 1$
 $() G_i \dots f + 2j \dots k - j \dots j$
 $0 \leq j < k$
 $() G_i \dots f + 2k - 1 \dots G_0 \oplus G_1 \dots f + 1 \dots k$

Note that the condition (a) of Theorem 1 is equivalent to that for any f' and k' such that $f' + 2k' \leq f + 2k$, G_i is f' -fault k' -disjoint path coverable. In this paper, we are concerned with a construction of f -fault many-to-many k -DPC of a graph G such that $f + 2k \leq \delta(G) - 1$.

Theorem 2. $k \geq 1 \dots f \geq 0, \dots G_i \dots n \dots i = 0, 1, 2, 3$
 $() G_i \dots f + 2j \dots k - j \dots j$
 $0 \leq j < k$
 $() G_i \dots f + 2k - 1 \dots H_0 \oplus H_1 \dots f \dots k + 1 \dots H_0 = G_0 \oplus G_1 \dots H_1 = G_2 \oplus G_3$

By applying the above two theorems to interconnection graphs, we will show that all of recursive circulant $G(2^m, 4)$, twisted cube TQ_m , and crossed cube CQ_m of degree m are f -fault many-to-many k -disjoint path coverable for every $k \geq 1$ and $f \geq 0$ such that $f + 2k \leq m - 1$.

Even when there are $p (< k)$ sources such that each source is identical with its corresponding sink, that is, when $s_i = t_i$ for all $1 \leq i \leq p$ and $S' \cap T' = \emptyset$, where $S' = \{s_{p+1}, \dots, s_k\}$ and $T' = \{t_{p+1}, \dots, t_k\}$, we can construct f -fault many-to-many k -DPC as follows: (a) we first define $P_i = (s_i)$, $1 \leq i \leq p$, a path with one vertex, and then (b) regarding them as virtual faulty vertices, find $f + p$ -fault many-to-many $k - p$ -DPC. Consequently, Proposition 3 can be extended so that adjacent edges are allowed.

2 Preliminaries

Let us consider fault-hamiltonicity of $G_0 \oplus G_1$. The following five lemmas are useful for our purpose. The proofs for them are omitted due to space limit.

Lemma 1. $f \geq 0, G_i \dots f \dots f + 1$
 $i = 0, 1, G_0 \oplus G_1 \dots f$
 $f + 1$

Lemma 2. $f \geq 2, G_i \dots f \dots f + 1$
 $i = 0, 1, G_0 \oplus G_1 \dots f + 1$

Lemma 3. $f = 0, 1, G_i \dots f \dots f + 1$
 $i = 0, 1, G_0 \oplus G_1 \dots f + 1$
 $s \dots t, s \dots t$

Lemma 4. $f \geq 1, G_i \dots f \dots f + 1$
 $i = 0, 1, G_0 \oplus G_1 \dots f + 2$

Lemma 5. $G \dots \delta \dots \delta \geq 3 \dots G \dots \delta - 3$
 $\dots \delta - 2 \dots G \times K_2 \dots \delta - 2$
 $\dots \delta - 1$

For a vertex v in $G_0 \oplus G_1$, we denote by \bar{v} the vertex adjacent to v which is in a component different from the component in which v is contained. We denote by U the set of terminals, the set of sources and sinks $S \cup T$, and denote by F the set of faulty elements.

Definition 1. $v \dots G_0 \oplus G_1 \dots$ free $v \notin F \dots v \notin U$
 $(v, w) \dots$ free $v \dots w \dots (v, w) \notin F$

Definition 2. free bridge $v \dots (v, \bar{v})$
 $\bar{v} \dots (v, \bar{v}) \notin F \dots (v, w, \bar{w})$
 $w \neq \bar{v}, (v, w) \notin F, (w, \bar{w})$

Lemma 6. $G_0 \oplus G_1 \dots k \dots f$
 $f + 2k \leq \Delta - 1, \Delta \dots G_0 \oplus G_1$
 $() \dots w \dots G_0 \oplus G_1 \dots w$

- () $W_l = \{w_1, w_2, \dots, w_l\}$ G_0 $l \leq 2k$
- l $w_i, 1 \leq i \leq l$
- () w_1 G_1 $W_l \setminus w_1 = \{w_2, \dots, w_l\}$
- G_0 $l \leq 2k$ l $w_i,$
- $1 \leq i \leq l$

There are at least Δ candidates for a free bridge of w , and at most $f + 2k - 1$ elements (f faulty elements and $2k - 1$ terminals other than w) can “block” the candidates. Since each element block at most one candidate, there are at least $\Delta - (f + 2k - 1) \geq 2$ nonblocked candidates, and thus (a) is proved. We prove (b) by induction on l . Before going on, we need some definitions. We call vertices v and \bar{v} and an edge joining them collectively a \dots of v . When (v, \bar{v}) (resp. (v, w, \bar{w})) is the free bridge of v , we say that the free bridge \dots a column of v (resp. two columns of v and w). We are to construct free bridges for W_l satisfying a condition that the number of occupied columns $c(l)$ is less than or equal to $f(l) + t(l)$, where $f(l)$ and $t(l)$ are the numbers of faulty elements and terminals contained in the $c(l)$ occupied columns, respectively. When $l = 1$, there exists a free bridge which satisfies the condition. Assume that there exist pairwise disjoint free bridges for $W_{l-1} = W \setminus w_l$ satisfying the condition. If (w_l, \bar{w}_l) is the free bridge of w_l , we are done. Suppose otherwise. There are Δ candidates for a free bridge, and the number of blocking elements is at most $c(l - 1)$ plus the number of terminals and faulty elements which are not contained in the $c(l - 1)$ occupied columns. Thus, the number of blocking elements is at most $f + 2k - 1$, which implies the existence of pairwise disjoint free bridges for W_l . Obviously, $c(l) = c(l - 1) + 2$ and $f(l) + t(l) \geq f(l - 1) + t(l - 1) + 2$, and thus it satisfies the condition.

Now, let us prove (c). If (w_1, \bar{w}_1) is the free bridge of w_1 , it occupies one column. If (w_1, x, \bar{x}) is the free bridge of w_1 and \bar{w}_1 is not a terminal of which we are to find a free bridge, it occupies two columns. For these cases, in the same way as (b), we can construct pairwise disjoint free bridges satisfying the above condition. When (w_1, x, \bar{x}) is the free bridge of w_1 and $\bar{w}_1 \in W_l$, letting $w_2 = \bar{w}_1$ without loss of generality, we first find pairwise disjoint free bridges of w_1 and w_2 . They occupy three columns, that is, $c(2) = 3$. We proceed to construct free bridges with a relaxed condition that $c(l) \leq f(l) + t(l) + 1$. This relaxation does not cause a problem since the number of blocking elements is at most $f + 2k$, still less than the number of candidates for a free bridge, Δ . □

According to the proof of Lemma 6 (a) and (b), we have at least two choices when we find free bridges of terminals contained in one component.

If G_i satisfies the conditions of Theorem 1 or 2, then $f + 2k \leq \delta - 1$, where $\delta = \min_i \delta(G_i)$. Concerned with Theorem 1, free bridges of type Lemma 6 (b) and (c) exist in $G_0 \oplus G_1$ since $(f + 1) + 2k \leq \delta(G_0 \oplus G_1) - 1$. Concerned with Theorem 2, free bridges of the two types also exist in $H_0 \oplus H_1$ since $f + 2(k + 1) \leq \delta(H_0 \oplus H_1) - 1$.

3 Construction of Many-to-Many DPC

In this section, we will prove the main theorems. First of all, we will develop five basic procedures for constructing many-to-many disjoint path covers. They play a significant role in proving the theorems.

3.1 Five Basic Procedures

In a graph $C_0 \oplus C_1$ with two components C_0 and C_1 , we are to define some notation. When we are concerned with Theorem 1, C_0 and C_1 correspond to G_0 and G_1 , respectively. When we are concerned with Theorem 2, C_0 and C_1 correspond to H_0 and H_1 , respectively. We denote by V_0 and V_1 the sets of vertices in C_0 and C_1 , respectively. We let F_0 and F_1 be the sets of faulty elements in C_0 and C_1 , respectively, and let F_2 be the set of faulty edges joining vertices in C_0 and vertices in C_1 . Let $f_i = |F_i|$ for $i = 0, 1, 2$.

We denote by R the set of source-sink pairs in $C_0 \oplus C_1$. We also denote by k_i the number of source-sink pairs in C_i , $i = 0, 1$, and by k_2 the number of source-sink pairs between C_0 and C_1 . Without loss of generality, we assume that $k_0 \geq k_1$. We let $I_0 = \{1, 2, \dots, k_0\}$, $I_2 = \{k_0 + 1, k_0 + 2, \dots, k_0 + k_2\}$, and $I_1 = \{k_0 + k_2 + 1, k_0 + k_2 + 2, \dots, k_0 + k_2 + k_1\}$. We assume that $\{s_j, t_j | j \in I_0\} \cup \{s_j | j \in I_2\} \subseteq V_0$ and $\{s_j, t_j | j \in I_1\} \cup \{t_j | j \in I_2\} \subseteq V_1$. Among the k_2 sources s_j 's, $j \in I_2$, we assume that the free bridges of k'_2 sources are of length one and the free bridges of $k''_2 (= k_2 - k'_2)$ sources are of length two.

First three procedures DPC-A, DPC-B, and DPC-C are applicable when $k_0 \geq 1$, and the last two procedures DPC-D and DPC-E are applicable when $k_2 = |R|$ (equivalently, $k_0 = k_1 = 0$). We denote by $H[v, w | G, F]$ a hamiltonian path in $G \setminus F$ joining a pair of fault-free vertices v and w in a graph G with a set F of faulty elements.

When we find a k -DPC or a hamiltonian path, sometimes we regard some fault-free vertices and/or edges as faulty elements. They are called \dots faults. For example, in step 2 of Procedure DPC-A, F' is the set of virtual vertex faults, and in step 2 of DPC-C, (s_2, s_1) in F' is a virtual edge fault.

Procedure DPC-A $(C_0 \oplus C_1, R, F)$ _____

UNDER the condition of $1 \leq k_0 < |R|$.

1. Find pairwise disjoint free bridges $B_{s_j} = (s_j, \dots, s'_j)$ of s_j for all $j \in I_2$.
2. Find k_0 -DPC $[\{(s_j, t_j) | j \in I_0\} | C_0, F_0 \cup F']$, where $F' = V_0 \cap \bigcup_{j \in I_2} V(B_{s_j})$.
3. Find $k_1 + k_2$ -DPC $[\{(s'_j, t_j) | j \in I_2\} \cup \{(s_j, t_j) | j \in I_1\} | C_1, F_1]$.
4. Merge the two DPC's with the free bridges.

Procedure DPC-B $(C_0 \oplus C_1, R, F)$ _____

UNDER the condition of $k_0 = |R|$.

1. Let s_1 and t_1 be a pair such that $|X_1| \leq |X_j|$ for all $j \in I_0$, where $X_j = V_0 \cap \{V(B_{s_j}) \cup V(B_{t_j})\}$. Let $B_{s_1} = (s_1, \dots, s'_1)$, $B_{t_1} = (t_1, \dots, t'_1)$.

2. Find $k_0 - 1$ -DPC $[\{(s_j, t_j) | j \in I_0 \setminus \{1\}\} | C_0, F_0 \cup X_1]$.
3. Find $H[s'_1, t'_1 | C_1, F_1]$.
4. Merge the $k_0 - 1$ -DPC and hamiltonian path with the free bridges.

Keep in mind that under the condition of procedure DPC-C below, for every $s_j, j \in I_2, \bar{s}_j = t_{j'}$ for some $j' \in I_2$, and thus for every other fault-free vertex v in $G_0, (v, \bar{v})$ is the free bridge of v .

Procedure DPC-C $(C_0 \oplus C_1, R, F)$ _____

UNDER the condition that $k_0 \geq 1, k_1 = 0, k'_2 = 0$, and all the faulty elements are contained in C_0 .

1. When $k_0 \geq 2$, find pairwise disjoint free bridges $B_{t_2} = (t_2, t'_2), B_{s_j} = (s_j, s'_j)$ and $B_{t_j} = (t_j, t'_j)$ for all $j \in I_0 \setminus \{1, 2\}$, and $B_{s_j} = (s_j, \dots, s'_j)$ for all $j \in I_2$. When $k_0 = 1$, find pairwise disjoint free bridges $B_{s_j} = (s_j, \dots, s'_j)$ for all $j \in I_2 \setminus 2$.
2. Find $H[s_2, t_1 | C_0, F_0 \cup F']$, where $F' = V_0 \cap [B_{t_2} \cup \bigcup_{j \in I_0 \setminus \{1, 2\}} (V(B_{s_j}) \cup V(B_{t_j})) \cup \bigcup_{j \in I_2} V(B_{s_j})]$ if $k_0 \geq 2$; $F' = \{(s_2, s_1)\} \cup (V_0 \cap \bigcup_{j \in I_2 \setminus 2} V(B_{s_j}))$ otherwise. Let the hamiltonian path be $(s_2, Q_1, z, s_1, Q_2, t_1)$.
3. Let $u = t'_2$ if $k_0 \geq 2$; otherwise, $u = t_2$. Find $k_0 + k_2 - 1$ -DPC $[\{\bar{z}, u\} \cup \{(s'_j, t'_j) | j \in I_0 \setminus \{1, 2\}\} \cup \{(s'_j, t_j) | j \in I_2 \setminus 2\} | C_1, \emptyset]$.
4. Merge the hamiltonian path and $k_0 + k_2 - 1$ -DPC with the free bridges and the edge (z, \bar{z}) . Discard the edge (z, s_1) .

Procedures DPC-D and DPC-E are concerned with the case of $k_2 = |R|$. Without loss of generality, we assume that $f_0 \geq f_1$. This assumption does not conflict with the assumption of $k_0 \geq k_1$.

Procedure DPC-D $(C_0 \oplus C_1, R, F)$ _____

UNDER the condition that $k_2 = |R| (k_0 = k_1 = 0)$.

1. If $k'_2 \geq 1$, we assume that (s_1, \bar{s}_1) is not the free bridge of s_1 . Find pairwise disjoint free bridges $B_{t_1} = (t_1, \dots, t'_1)$ and $B_{s_j} = (s_j, \dots, s'_j)$ for all $j \in I_2 \setminus 1$.
2. Find $H[s_1, t'_1 | C_0, F_0 \cup F']$, where $F' = V_0 \cap \bigcup_{j \in I_2 \setminus 1} V(B_{s_j})$.
3. Find $k_2 - 1$ -DPC $[\{(s'_j, t_j) | j \in I_2 \setminus 1\} | C_1, F_1 \cup F'']$, where $F'' = V_1 \cap B_{t_1}$.
4. Merge the hamiltonian path and the $k_2 - 1$ -DPC with the free bridges.

Observe that under the condition of procedure DPC-E below, for every source s_j in $G_0, \bar{s}_j = t_{j'}$ for some $j' \in I_2$, and thus for any free vertex v in $G_0, (v, \bar{v})$ is a free edge.

Procedure DPC-E $(C_0 \oplus C_1, R, F)$ _____

UNDER the condition that $k_2 = |R|, k'_2 = 0$, and all the faulty elements are contained in C_0 .

1. Find pairwise disjoint free bridges $B_{t_1} = (t_1, \dots, t'_1)$ and $B_{s_j} = (s_j, \dots, s'_j)$ for all $j \in I_2 \setminus \{1, 2\}$.
2. Find $H[s_2, t'_1 | C_0, F_0 \cup F']$, where $F' = \{(s_1, s_2)\} \cup (V_0 \cap \bigcup_{j \in I_2 \setminus \{1, 2\}} V(B_{s_j}))$. Let the hamiltonian path be $(s_2, \dots, z, s_1, \dots, t'_1)$.
3. Find $k_2 - 1$ -DPC $\{(\bar{z}, t_2)\} \cup \{(s'_j, t_j) | j \in I_2 \setminus \{1, 2\}\} | C_1, F''$, where $F'' = V_1 \cap V(B_{t_1})$.
4. Merge the hamiltonian path and the $k_2 - 1$ -DPC with the free bridges. Discard the edge (s_1, z) .

3.2 Proof of Theorem 1

For $k = 1$ and $f \geq 2$, the theorem is exactly the same as Lemma 2. We assume that

$$k \geq 2, f_0 + f_1 + f_2 \leq f + 1, \text{ and } k_0 + k_1 + k_2 = k.$$

Lemmas 7, 8, and 9 are concerned with $k_0 \geq 1$, and Lemmas 10 and 11 are concerned with $k_2 = k$.

Lemma 7. $1 \leq k_0 < k, f_0 = f + 1, k_1 = 0, k'_2 = 0$ $(G_0 \oplus G_1, R, F)$

The existence of pairwise disjoint free bridges in step 1 is due to Lemma 6 (b). Unless $f_0 = f + 1, k_1 = 0,$ and $k'_2 = 0, G_0$ is $f_0 + k'_2 + 2k'_2$ -fault k_0 -disjoint path coverable since $2k_0 + f_0 + k'_2 + 2k'_2 \leq 2k + f,$ and thus there exists a k_0 -DPC in step 2. Similarly, G_1 is f_1 -fault $k_1 + k_2$ -disjoint path coverable since $2k_1 + 2k_2 + f_1 \leq 2k + f.$ This completes the proof of the lemma. \square

Lemma 8. $k_0 = k, f_0 = f + 1 (k_1 = 0, k'_2 = 0)$ $(G_0 \oplus G_1, R, F)$ $f + 1$

To prove the existence of a $k - 1$ -DPC in step 2, we will show that $f_0 + |X_1| \leq f + 2.$ When $|X_1| = 2,$ the inequality holds true unless $f_0 = f + 1.$ When $|X_1| = 3,$ the number $f_1 + f_2$ of faulty elements in G_1 or between G_0 and G_1 is at least $k (\geq 2),$ and thus $f_0 + 3 \leq f_0 + f_1 + f_2 + 1 \leq f + 2.$ When $|X_1| = 4,$ analogously to the previous case, $f_0 + 4 \leq f_0 + f_1 + f_2 < f + 2$ since $f_1 + f_2 \geq 2k.$ The existence of a hamiltonian path joining s'_1 and t'_1 is due to the fact that $f_1 \leq f + 2k - 2.$ \square

Lemma 9. $k_0 \geq 1, f_0 = f + 1, k_1 = 0, k'_2 = 0$ $(G_0 \oplus G_1, R, F)$ $f + 1, k$

Whether $k_0 \geq 2$ or not, it holds true that $f_0 + |F'| \leq f + 1 + 2(k - 2) + 1 = f + 2k - 2,$ which implies the existence of a hamiltonian path in step 2. By the construction, (z, \bar{z}) is the free bridge of $z.$ Note that $z \neq s_2$ when $k_0 = 1.$ The existence of a $k - 1$ -DPC in step 3 is straightforward. \square

Lemma 10. $k_2 = k, f_0 = f + 1, k'_2 = 0$ $(G_0 \oplus G_1, R, F)$

The existence of pairwise disjoint free bridges is due to Lemma 6(c). To prove the existence of the hamiltonian path, we will show that $f_0 + |F'| \leq f + 2k - 2$. When $k_2'' \geq 1$, $f_0 + |F'| = f_0 + 2(k_2'' - 1) + k_2' \leq f + 2k - 2$ unless $f_0 = f + 1$ and $k_2' = 0$. When $k_2'' = 0$, $f_0 + |F'| = f_0 + k_2' - 1 \leq f + 2k - 2$. The existence of $k_2 - 1$ -DPC in step 3 is due to that $f_1 + |F''| \leq f + 2$. Note that the assumption that $f_0 \geq f_1$ implies that $f_1 < f + 1$. \square

Lemma 11. $(G_0 \oplus G_1, R, F)$ is $(f + 1, k)$ -DPC if and only if $k_2 = k$, $f_0 = f + 1$, and $k_2' = 0$. \square

The existence of the hamiltonian path is due to the fact that $f_0 + |F'| = f_0 + 2(k_2 - 2) + 1 \leq f + 2k - 2$. Note that z is different from s_1 and s_2 , and thus (z, \bar{z}) is a free edge. The existence of the $k_2 - 1$ -DPC is straightforward. \square

Consequently, the proof of Theorem 1 is completed. From Theorem 1 and Lemma 4, the following corollary is immediate.

Corollary 1. Let $k \geq 2$ and $f \geq 0$, or $k = 1$ and $f \geq 2$. Let G_i be an n -vertex $(f + 2j + 1, k - j)$ -DPC for $i = 0, 1$. Then $(G_0 \oplus G_1, R, F)$ is $(f + 2k, k)$ -DPC if and only if $j, 0 \leq j < k$.

3.3 Proof of Theorem 2 for $k \geq 2$ and $f \geq 0$ or for $k = 1$ and $f \geq 2$

Corollary 1 implies that $H_i, i = 0, 1$, is $f + 2j + 1$ -fault many-to-many $k - j$ -disjoint path coverable for every $j, 0 \leq j < k$, and that H_i is $f + 2k$ -fault hamiltonian. In this subsection, by utilizing mainly these properties of H_i , we are to prove Theorem 2 for $k \geq 2$ and $f \geq 0$ or for $k = 1$ and $f \geq 2$. We assume that

$$f_0 + f_1 + f_2 \leq f \text{ and } k_0 + k_1 + k_2 = k + 1.$$

Similarly to the proof of Theorem 1, Lemmas 12, 13, and 14 are concerned with $k_0 \geq 1$, and Lemmas 15 and 17 are concerned with $k_2 = k + 1$.

Lemma 12. $(H_0 \oplus H_1, R, F)$ is $(f, k + 1)$ -DPC if and only if $f_0 = f, k_1 = 0$, and $k_2' = 0$.

Unless $f_0 = f, k_1 = 0$, and $k_2' = 0, H_0$ is $f_0 + k_2' + 2k_2''$ -fault k_0 -disjoint path coverable since $2k_0 + f_0 + k_2' + 2k_2'' \leq 2k + f + 1$, and thus there exists a k_0 -DPC in step 2. Similarly, H_1 is f_1 -fault $k_1 + k_2$ -disjoint path coverable since $2k_1 + 2k_2 + f_1 \leq 2k + f + 1$. \square

Lemma 13. $(H_0 \oplus H_1, R, F)$ is $(f, k + 1)$ -DPC if and only if $f_0 = f (k_1 = 0)$ and $k_2' = 0$.

To prove the existence of a k -DPC in step 2, we will show that $f_0 + |X_1| \leq f + 1$. When $|X_1| = 2$, the inequality holds true unless $f_0 = f$. When $|X_1| = 3$, it holds true that $f_1 + f_2 \geq k + 1$, and thus $f_0 + 3 \leq f_0 + f_1 + f_2 + 1 \leq f + 1$. When $|X_1| = 4, f_0 + 4 \leq f_0 + f_1 + f_2 < f + 1$ since $f_1 + f_2 \geq 2(k + 1)$. Obviously, there exists a hamiltonian path in H_1 joining s_1' and t_1' . \square

Lemma 14. $k_0 \geq 1, f_0 = f, k_1 = 0, k'_2 = 0, (H_0 \oplus H_1, R, F) \dots f \dots k + 1$

There exists a hamiltonian path in H_0 joining s_2 and t_1 since $f_0 + |F'| \leq f + 2(k - 1) + 1 = f + 2k - 1$. The existence of a k -DPC is straightforward. \square

Hereafter in this subsection, $k_2 = k + 1$ ($k_0 = k_1 = 0$). Due to Lemma 6(a) and Remark 2, we assume that F'' defined in step 3 of Procedures DPC-D and DPC-E is a subset of $V(G_2)$ or $V(G_3)$. That is, $F'' \cap V(G_2) \neq \emptyset$ if and only if $F'' \cap V(G_3) = \emptyset$.

Lemma 15. $k_2 = k + 1, (H_0 \oplus H_1, R, F) \dots f \dots k + 1 \dots f_0 = f \dots k'_2 = 0$

To prove the existence of a hamiltonian path in H_0 , we will show that $f_0 + |F'| \leq f + 2k - 1$. When $k'_2 \geq 1, f_0 + |F'| = f_0 + 2(k'_2 - 1) + k'_2 \leq f + 2k - 1$ unless $f_0 = f$ and $k'_2 = 0$. When $k'_2 = 0, f_0 + |F'| = f_0 + k'_2 - 1 \leq f + 2k - 1$. Now, let us consider the existence of a $k_2 - 1$ -DPC in step 3. When $f \geq 1$ or $|F''| = 1$, there exists a $k_2 - 1$ -DPC in H_1 since $f_1 + |F''| \leq f + 1$. Note that from the assumption of $f_0 \geq f_1$, if $f \geq 1$, then $f_1 < f$. When $f = 0$ and $|F''| = 2$ ($k \geq 2$ by the assumption), the existence of a $k_2 - 1$ -DPC is due to the following Lemma 16. \square

The proof of Lemma 16 is omitted. Of course, Lemma 16 does not say that $G_0 \oplus G_1$ is 2-fault many-to-many k -disjoint path coverable.

Lemma 16. $k \geq 2, G_i \dots n, \dots i = 0, 1 \dots G_i \dots 2j \dots k - j, \dots j, 0 \leq j < k, \dots G_i \dots 2k - 1, \dots G_0 \oplus G_1 \dots k$

Lemma 17. $k_2 = k + 1, f_0 = f, k'_2 = 0, (H_0 \oplus H_1, R, F) \dots f \dots k + 1$

There exists a hamiltonian path in H_0 joining s_2 and t'_1 since $f_0 + |F'| = f_0 + 2(k_2 - 2) + 1 = f + 2k - 1$. When $f \geq 1$, there exists a $k_2 - 1$ -DPC in H_1 since $|F''| = 2 \leq f + 1$. When $f = 0$ (and $|F''| = 2$), the existence of a $k_2 - 1$ -DPC is due to Lemma 16. \square

3.4 Proof of Theorem 2 for $k = 1$ and $f = 0, 1$

In $H_0 \oplus H_1, H_0$ and H_1 are called components and $G_i, 0 \leq i \leq 3$, are called \dots . Contrary to the proof given in the previous subsection, we can not employ Corollary 1. Instead, Lemma 1 and 3 are utilized repeatedly in this subsection. We denote by \hat{v} the vertex which is adjacent to v and contained in the same component with v and in a different subcomponent from v . Lemmas 18, 19, and 20 are concerned with $k_0 \geq 1$. It is assumed that $k_0 \geq k_1$. All the proofs of lemmas in this subsection are omitted.

Lemma 18. $(k_0 = 1, f = 2, f_0 = f, k_1 = 0, k'_2 = 0)$

Lemma 19. $(k_0 = 2, f = 2, f_0 = f, k_1 = 0, k'_2 = 0)$

Lemma 20. $(k_0 \geq 1, f_0 = f, k_1 = 0, k'_2 = 0, f = 2)$

Now, let us consider the case when $k_2 = 2$ ($k_0 = k_1 = 0$). We assume that $f_0 \geq f_1$. Then, $f_1 = 0$. We denote by $l_{i,j}$ the number of edges joining vertices in G_i and $G_j, i \neq j$. Observe that $l_{0,1} = n, l_{0,2} + l_{0,3} = n, l_{0,2} = l_{1,3}$, and $l_{0,3} = l_{1,2}$. Note that $n \geq f + 4$ since each G_i is $f + 1$ -fault hamiltonian.

Lemma 21. $(k_2 = 2, f = 2, f_0 = f, k'_2 = 0)$

Lemma 22. $(k_2 = 2, f = 0, (s_1, t_1) \in E, k'_2 = 1, f = 2)$

Lemma 23. $(k_2 = 2, f_0 = f, k'_2 = 0, f = 2)$

At last, the proof of Theorem 2 is completed. From Theorem 2, we have the following corollary.

Corollary 2. $(k \geq 1, f \geq 0, G_i, n, H_0 \oplus H_1, f + 2j, k + 1 - j, j, 0 \leq j < k, H_0 = G_0 \oplus G_1, H_1 = G_2 \oplus G_3)$

4 Hypercube-Like Interconnection Networks

A graph G is called (k, f) -many-to-many disjoint path coverable if for any $k \geq 1$ and $f \geq 0$ such that $f + 2k \leq \delta(G) - 1, G$ is f -fault many-to-many k -disjoint path coverable.

4.1 Recursive Circulants $G(2^m, 4)$

$G(2^m, 4)$ is an m -regular graph with 2^m vertices. According to the recursive structure of recursive circulants[10], we can observe that $G(2^m, 4)$ is isomorphic to $G(2^{m-2}, 4) \times K_2 \oplus_M G(2^{m-2}, 4) \times K_2$ for some permutation M . Obviously, $G(2^{m-2}, 4) \times K_2$ is isomorphic to $G(2^{m-2}, 4) \oplus_{M'} G(2^{m-2}, 4)$ for some M' . Fault-hamiltonicity of $G(2^m, 4)$ was studied in [11]. By utilizing Lemma 5, we can also obtain fault-hamiltonicity of $G(2^m, 4) \times K_2$.

Lemma 24. $(G(2^m, 4), m \geq 3, m - 3, m - 2, G(2^m, 4) \times K_2, m \geq 3, m - 2, m - 1)$

Theorem 3. $G(2^m, 4), m \geq 3,$

The proof is by induction on m . For $m = 3, 4$, the theorem holds true by Lemma 24. For $m \geq 5$, from Corollary 2 and Lemma 24, the theorem follows immediately. \square

4.2 Twisted Cube TQ_m , Crossed Cube CQ_m

Originally, twisted cube TQ_m is defined for odd m . We let $TQ_m = TQ_{m-1} \times K_2$ for even m . Then, TQ_m is isomorphic to $TQ_{m-1} \oplus_M TQ_{m-1}$ for some M . Also, crossed cube CQ_m is isomorphic to $CQ_{m-1} \oplus_{M'} CQ_{m-1}$ for some M' . Both TQ_m and CQ_m are m -regular graphs with 2^m vertices. Fault-hamiltonicity of them were studied in the literature.

Lemma 25. () $TQ_m, m \geq 3, \dots, m-3$ $\dots \dots \dots m-2$
 () $CQ_m, m \geq 3, \dots, m-3$ $\dots \dots \dots m-2$

From Lemma 5, Corollary 2, and Lemma 25, we have the following theorem.

Theorem 4. $TQ_m \dots CQ_m, m \geq 3, \dots \dots \dots$

References

1. J.A. Bondy and U.S.R. Murty, Graph Theory with Applications, 5th printing, American Elsevier Publishing Co., Inc., 1976.
2. C.C. Chen and J. Chen, "Nearly optimal one-to-many parallel routing in star networks," *IEEE Transactions on Parallel and Distributed Systems* **8(12)**, pp. 1196-1202, 1997.
3. S. Gao, B. Novick and K. Qiu, "From hall's matching theorem to optimal routing on hypercubes," *Journal of Combinatorial Theory, Series B.* **74**, pp. 291-301, 1998.
4. Q.P. Gu and S. Peng, "Cluster fault-tolerant routing in star graphs," *Networks* **35(1)**, pp. 83-90, 2000.
5. W.T. Huang, M.Y. Lin, J.M. Tan, and L.H. Hsu, "Fault-tolerant ring embedding in faulty crossed cubes," in *Proc. SCI'2000*, pp. 97-102, 2000.
6. W.T. Huang, J.M. Tan, C.N. Huang, L.H. Hsu, "Fault-tolerant hamiltonicity of twisted cubes," *J. Parallel Distrib. Comput.* **62**, pp. 591-604, 2002.
7. S. Madhavapeddy and I.H. Sudborough, "A topological property of hypercubes: node disjoint paths," in *Proc. of the 2th IEEE Symposium on Parallel and Distributed Processing*, pp. 532-539, 1990.
8. J.-H. Park, "One-to-one disjoint path covers in recursive circulants," *Journal of KISS* **30(12)**, pp. 691-698, 2003 (in Korean).
9. J.-H. Park, "One-to-many disjoint path covers in a graph with faulty elements," in *Proc. International Computing and Combinatorics Conference COCOON2004*, pp. 392-401, 2004.
10. J.-H. Park and K.Y. Chwa, "Recursive circulants and their embeddings among hypercubes," *Theoretical Computer Science* **244**, pp. 35-62, 2000.
11. C.-H. Tsai, J.J.M. Tan, Y.-C. Chuang, and L.-H. Hsu, "Fault-free cycles and links in faulty recursive circulant graphs," in *Proc. of Workshop on Algorithms and Theory of Computation ICS2000*, pp. 74-77, 2000.
12. C.-H. Tsai, J.J.M. Tan, and L.-H. Hsu, "The super-connected property of recursive circulant graphs," *Inform. Proc. Lett.* **91(6)**, pp. 293-298, 2004.

An $O(n \log n)$ -Time Algorithm for the Maximum Constrained Agreement Subtree Problem for Binary Trees

Zeshan Peng and Hingfung Ting

Department of Computer Science,
The University of Hong Kong, Hong Kong,
{zspeng, hfting}@cs.hku.hk

Abstract. This paper introduces the maximum constrained agreement subtree problem, which is a generalization of the classical maximum agreement subtree problem. This new problem is motivated by the understood constraint when we compare the evolutionary trees. We give an $O(n \log n)$ -time algorithm for solving the problem when the input trees are binary. The time complexity of our algorithm matches the fastest known algorithm for the maximum agreement subtree problem for binary trees.

1 Introduction

A fundamental problem in Biology is to recover the evolutionary relationship of the species in nature. One model for capturing this relationship is the evolutionary tree [5, 6], which is a rooted tree with its leaves labeled by a unique species. Different theories capture different kinds of evolutionary relationship and induce different evolutionary trees. To compare different theories and to find out how much these theories are in common, we compare the corresponding evolutionary trees and find some consensus of these trees.

One of the most successful approaches for finding consensus of different evolutionary trees is to construct their maximum agreement subtree. Roughly speaking, an *agreement subtree* of two evolutionary trees is an evolutionary tree which is also a topology subtree of the two given trees. A *maximum agreement subtree* (MAST) is one with the largest possible number of leaves. There are many algorithms proposed for finding the MAST of two trees (e.g., [2, 3, 4, 10, 11, 12]). The fastest algorithm for the problem is given by Kao, Lam, Sung and Ting [8, 9]; it runs in $O(n^{1.5})$ time where n is the total number of leaves. Kao [7] showed that the time complexity can be reduced to $O(n \log^2 n)$ if the degrees of the input trees are bounded. Cole, Farach, Hariharan, Przytycka and Throup [1] showed that the running time can be further reduced to $O(n \log n)$ when the two input trees are binary. Note that this is the most important special case of the problem because in nature, it is rare that more than two species evolved simultaneously from one ancestor.

In this paper, we introduce the *maximum constrained agreement subtree* problem, which is defined as follows:

Let S and T be two evolutionary trees, and P be an agreement subtree of S and T . Find the largest agreement subtree of S and T that contains P as a subtree. We say that this agreement subtree is the *maximum constrained agreement subtree* (MCAST) of S and T with respect to P .

Note that when P is the empty tree, our problem becomes the classical maximum agreement subtree problem.

Following is the motivation of our problem. After decades of research, the evolutionary relationship of many species is well understood. Any evolutionary tree having these species should be consistent with this commonly accepted relationship. With this additional constraint, MAST is not a good measure for comparing evolutionary trees. For example, consider the evolutionary trees S and T in Figure 1. Note that the maximum agreement subtree of S and T is large, and one would consider that the two trees are similar. However, the two trees agree on almost nothing if we insist that the agreement subtree must be consistent with the evolutionary relationship of e, f, h , which is given by P . In fact, if P is a correct relationship, then S and T infer different evolutionary relationship for all other species. For example, for the species a , S suggests that the least common ancestor of a and e is different from the least common ancestor of a and f , while T suggests they are the same. Note that the MCAST of S and T with respect to P is just P .

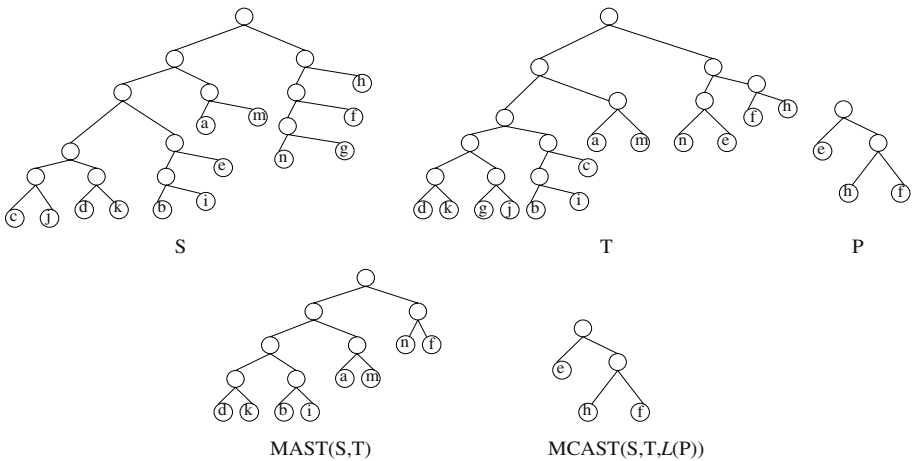


Fig. 1. Maximum agreement and maximum constrained agreement subtrees

In this paper, we give an algorithm for constructing a MCAST of S and T with respect to P . We focus on the most important special case when both S and T are binary trees. The idea of our algorithm is as follows. By some careful analysis, we observe that when P has more than two leaves, any MCAST of

S and T with respect to P can be decomposed into three subtrees where each subtree corresponds to a MCAST of some smaller trees. Furthermore, these smaller trees can be constructed easily. Hence, we can solve any instance of the MCAST problem by solving three smaller instances recursively. We do not have this nice structural property when P has fewer than two leaves. Note that when P has no leaf, then our problem becomes the classical MAST problem and we can solve them efficiently by using the algorithm of Cole [1]. When P has a single leaf, the MCAST and MAST can be very different (see Figure 2 for an example). However, based on a simple idea, we can reduce the problem to the MAST problem and thus can apply Cole’s algorithm for the case when P has one leaf.

Based on these observations and ideas, we design an $O(n \log n)$ -time algorithm for constructing a MCAST of S and T with respect to P . Our algorithm is simple; it has three recursive calls when P has more than one leaf. If P has one leaf, it calls Cole’s algorithm. Although our algorithm is simple, its correctness proof is not easy. It is based on a correct layout of the trees S and T and a careful analysis of some structural properties on this layout.

The paper is organized as follows. In Section 2, we formally define the MAST and the MCAST problem and state some facts that are useful to our discussion. In Section 3, we show that if we only P has only one leaf, then the MCAST problem can be reduced to the MAST problem easily. Section 4 contains the main result of this paper; we design and analysis of an $O(n \log n)$ time algorithm for the MCAST problem without any restriction. Section 5 gives the conclusion.

2 Definitions and Notations

An evolutionary tree S is a rooted tree with every leaf being labeled with a unique species. We let $\mathcal{L}(S)$ denote the set of species labeling S . When there is no confusion, we use the species to identify the leaves. For any two leaves a, b in $\mathcal{L}(S)$, we say a node c is an ancestor of a and b if it is the ancestor of a and b . It is the least common ancestor of a and b if for any other ancestor c' of a, b , c' is an ancestor of c . We use $\text{lca}_S(a, b)$ to denote the least common ancestor of a, b in S .

Let T be another evolutionary tree. We say that S and T are isomorphic if $\mathcal{L}(S) = \mathcal{L}(T)$ and there exists a bijection f from the nodes of S to the nodes of T such that

1. for any leaf a of S , $f(a)$ is the leaf in T that has the same label a , i.e., $f(a) = a$;
2. for any pair of leaves u, v of S , if w is the least common ancestor of u, v in S , then $f(w)$ is the least common ancestor of $f(u)$ and $f(v)$ in T ; i.e., $f(\text{lca}_S(a, b)) = \text{lca}_T(a, b)$.

We write $S = T$ if the two trees are label-preserving isomorphic. It is easy to check that the bijection f exists if and only if the following holds: any two pairs of leaves have the same least common ancestor in S if and only if they have the same least common ancestor in T . Thus, we have the following fact.

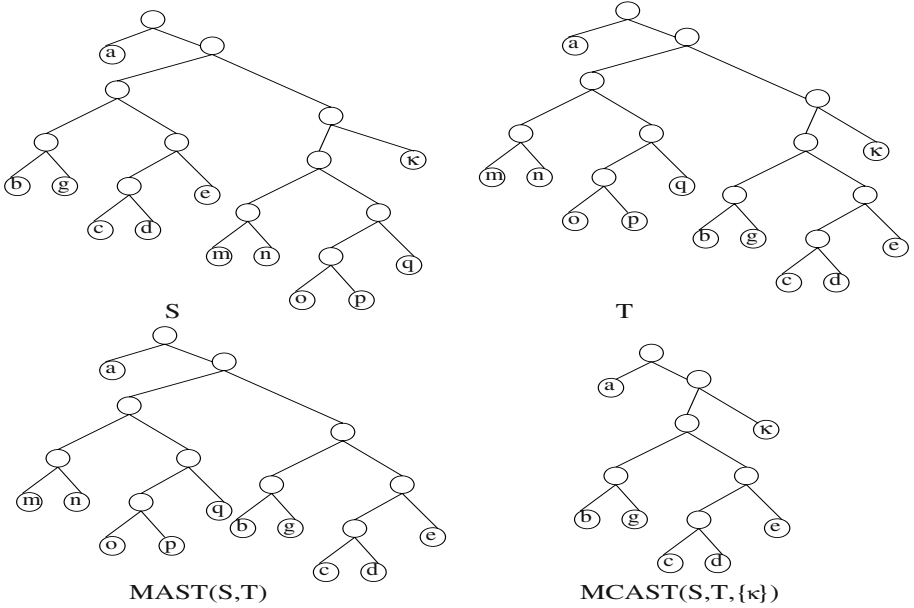


Fig. 2. The agreement subtrees with and without the leaf κ

$|\mathcal{L}(W)| > |\mathcal{L}(S)| + |\mathcal{L}(T)|$, any maximum agreement subtree R of S' and T' must include W . In other words, the role of W is equivalent to the role of κ in our problem. If we replace the subtree W in R by the leaf κ , we get a maximum constrained agreement subtree of S and T with respect to the κ .

The following lemma proves formally the correctness of this idea.

Lemma 1. S' T' κ T W R κ S
 $G = (\mathcal{L}(R) - \mathcal{L}(W)) \cup \{\kappa\}$ $G \in$
 $\text{MCAST}(S, T, \{\kappa\})$

Note that W is an agreement subtree of S' and T' and thus $|\mathcal{L}(R)| \geq |\mathcal{L}(W)| = n + 1$. On the other hand, $|\mathcal{L}(R) - \mathcal{L}(W)| \leq |\mathcal{L}(S)| + |\mathcal{L}(T)| = n$. Thus, R must have some leaf $\tau \in \mathcal{L}(W)$. Define $S'_{\tau \rightarrow \kappa}$ to be the tree obtained by replacing this leaf τ by κ in S' . Define $T'_{\tau \rightarrow \kappa}$ similarly. By the definition of agreement subtree, $S' \parallel_{\mathcal{L}(R)} T' = T' \parallel_{\mathcal{L}(R)} S'$ and this implies $S'_{\tau \rightarrow \kappa} \parallel_{(\mathcal{L}(R) - \{\tau\}) \cup \{\kappa\}} = T'_{\tau \rightarrow \kappa} \parallel_{(\mathcal{L}(R) - \{\tau\}) \cup \{\kappa\}}$. By Fact 2, we have

$$S'_{\tau \rightarrow \kappa} \parallel_{((\mathcal{L}(R) - \{\tau\}) \cup \{\kappa\}) - \mathcal{L}(W)} = T'_{\tau \rightarrow \kappa} \parallel_{((\mathcal{L}(R) - \{\tau\}) \cup \{\kappa\}) - \mathcal{L}(W)}$$

Note that $G = (\mathcal{L}(R) - \mathcal{L}(W)) \cup \{\kappa\} = (\mathcal{L}(R) - \{\tau\}) \cup \{\kappa\} - \mathcal{L}(W)$ and thus

$$S'_{\tau \rightarrow \kappa} \parallel_G = T'_{\tau \rightarrow \kappa} \parallel_G$$

Observe that $S = S'_{\tau \rightarrow \kappa} \parallel_{\mathcal{L}(S)}$ and $T = T'_{\tau \rightarrow \kappa} \parallel_{\mathcal{L}(T)}$. Furthermore, since $G \subseteq \mathcal{L}(S)$, $(S'_{\tau \rightarrow \kappa} \parallel_{\mathcal{L}(S)}) \parallel_G = S'_{\tau \rightarrow \kappa} \parallel_G$. Similarly, we have $(T'_{\tau \rightarrow \kappa} \parallel_{\mathcal{L}(T)}) \parallel_G = T'_{\tau \rightarrow \kappa} \parallel_G$. Altogether, we have

$$S \parallel_G = (S'_{\tau \rightarrow \kappa} \parallel_{\mathcal{L}(S)}) \parallel_G = S'_{\tau \rightarrow \kappa} \parallel_G = T'_{\tau \rightarrow \kappa} \parallel_G = (T'_{\tau \rightarrow \kappa} \parallel_{\mathcal{L}(T)}) \parallel_G = T \parallel_G.$$

Together with the fact that $\kappa \in G$, we conclude $G \in \text{CAST}(S, T, \{\kappa\})$.

Now, we show that $G \in \text{MCAST}(S, T, \{\kappa\})$. For any $L \in \text{CAST}(S, T, \{\kappa\})$, $(L - \{\kappa\}) \cup \mathcal{L}(W)$ corresponds to an agreement subtree of S' and T' and thus

$$|L| - 1 + |\mathcal{L}(W)| \leq |\mathcal{L}(R)|.$$

On the other hand,

$$|G| = |(\mathcal{L}(R) - \mathcal{L}(W)) \cup \{\kappa\}| \geq |\mathcal{L}(R)| - |\mathcal{L}(W)| + 1.$$

It follows that $|G| \geq |L|$ and G is no smaller than any label set in $\text{CAST}(S, T, \{\kappa\})$. Hence, $G \in \text{MCAST}(S, T, \{\kappa\})$.

Therefore, we can reduce the problem of finding a label set in $\text{MCAST}(S, T, K)$ to the problem of finding maximum agreement subtree when $|K| \leq 1$. By applying the algorithm of Cole [1995], we have a procedure for solving this special case in $O(n \log n)$ time. For ease of future referencing, we refer this procedure as $\text{SimpleMCAST}(S, T, K)$. In the next section, we will see how to use this procedure to solve the general problem.

4 The General Case

In this section, we describe a simple recursive procedure for finding a label set in $\text{MCAST}(S, T, K)$ when the size of K is general. First, we need some definitions.

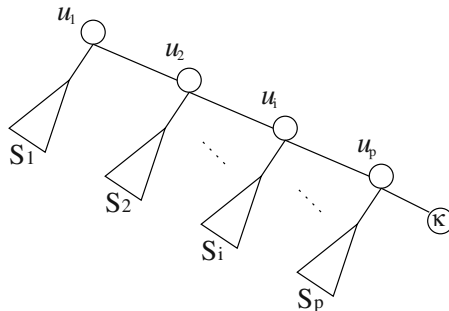


Fig. 3. The spine decomposition of S at κ

Let κ be any leaf of S . Let $\sigma = (u_1, u_2, \dots, u_p, \kappa)$ be the unique path from the root u_1 of S to κ . We say that S has the spine decomposition $\mathcal{D}(S_1, S_2, \dots, S_p, \kappa)$

at κ if along σ , $S_i(1 \leq i \leq p)$ is the subtree attached to u_i (see Figure 3). We call the path σ the κ -spine (or simply spine) and the S_i 's the κ -sidetrees (or simply sidetrees) of S . We call u_i the κ -parent of S_i , and we let $p(S_i)$ to denote S_i 's parent. We say that a sidetree S_h is *higher* than another sidetree S_k if $h < k$.

Following is our procedure for finding $\text{mCAST}(S, T, K)$. To simplify notation, we let $\mathcal{L}(X_1, X_2, \dots, X_k)$ to denote $\mathcal{L}(X_1) \cup \mathcal{L}(X_2) \cup \dots \cup \mathcal{L}(X_k)$.

procedure ConstMCAST(S, T, K)

1. if $|K| \leq 1$, return SimpleMCAST(S, T, K);
2. Let κ be a label in K ;
3. Let $\mathcal{D}(S_1, S_2, \dots, S_p, \kappa)$ and $\mathcal{D}(T_1, T_2, \dots, T_q, \kappa)$ be the spine decomposition of S and T at κ , respectively.
4. Find the highest sidetree S_i of S that contains some label in K ;
5. Find the highest sidetree T_j of T that contains some label in K ;
6. Let $S_{\text{top}} := S \parallel_{\mathcal{L}(S_1, S_2, \dots, S_{i-1}) \cup \{\kappa\}}$ and $S_{\text{bot}} := S \parallel_{\mathcal{L}(S_{i+1}, S_{i+2}, \dots, S_p) \cup \{\kappa\}}$;
7. Let $T_{\text{top}} := T \parallel_{\mathcal{L}(T_1, T_2, \dots, T_{j-1}) \cup \{\kappa\}}$ and $T_{\text{bot}} := T \parallel_{\mathcal{L}(T_{j+1}, T_{j+2}, \dots, T_q) \cup \{\kappa\}}$;
8. Recursively call the procedure as follows:
 $A := \text{ConstMCAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}}))$;
 $B := \text{ConstMCAST}(S_i, T_j, K \cap \mathcal{L}(S_i))$;
 $C := \text{ConstMCAST}(S_{\text{bot}}, T_{\text{bot}}, K \cap \mathcal{L}(S_{\text{bot}}))$;
9. return $A \cup B \cup C$.

The following lemma suggests that the label set returned by the procedure is indeed in $\text{CAST}(S, T, K)$.

Lemma 2.

- $A \in \text{CAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}}))$.
- $B \in \text{CAST}(S_i, T_j, K \cap \mathcal{L}(S_i))$.
- $C \in \text{CAST}(S_{\text{bot}}, T_{\text{bot}}, K \cap \mathcal{L}(S_{\text{bot}}))$

$$L = A \cup B \cup C \quad \dots \quad K \subseteq L \quad \dots \quad S \parallel_L = T \parallel_L \quad \dots \quad L \in \text{CAST}(S, T, K)$$

To show $K \subseteq L$, note that $\mathcal{L}(S_{\text{top}}) \cup \mathcal{L}(S_i) \cup \mathcal{L}(S_{\text{bot}}) = \mathcal{L}(S)$, and thus

$$K = K \cap \mathcal{L}(S) = (K \cap \mathcal{L}(S_{\text{top}})) \cup (K \cap \mathcal{L}(S_i)) \cup (K \cap \mathcal{L}(S_{\text{bot}})) \subseteq A \cup B \cup C = L.$$

To prove that $S \parallel_L = T \parallel_L$, it suffices to show that for any leaves a, b, c, d in L , $\text{lca}_{S \parallel_L}(a, b) = \text{lca}_{S \parallel_L}(c, d)$ if and only if $\text{lca}_{T \parallel_L}(a, b) = \text{lca}_{T \parallel_L}(c, d)$ (see Fact 1).

First, we consider the \implies direction. Suppose $\text{lca}_{S \parallel_L}(a, b) = \text{lca}_{S \parallel_L}(c, d) = u$. We consider two cases.

Case 1: u is a node in some sidetree S_ℓ where $1 \leq \ell \leq p$. Note that a, b, c, d are all in S_ℓ because their ancestor is in S_ℓ . Since $A \subseteq \mathcal{L}(S_1, S_2, \dots, S_{i-1}) \cup \{\kappa\}$, $B \subseteq \mathcal{L}(S_i)$ and $C \subseteq \mathcal{L}(S_{i+1}, \dots, S_p) \cup \{\kappa\}$, S_ℓ does not have two leaves from two different sets in $\{A, B, C\}$. In other words, $\{a, b, c, d\} \subseteq X$ where X is

either A , B , or C . Since $X \subseteq L$, by Fact 2, $\text{lca}_{S\|_L}(a, b) = \text{lca}_{S\|_X}(a, b)$ and $\text{lca}_{S\|_L}(c, d) = \text{lca}_{S\|_X}(c, d)$. Hence,

$$\text{lca}_{S\|_L}(a, b) = \text{lca}_{S\|_L}(c, d) \iff \text{lca}_{S\|_X}(a, b) = \text{lca}_{S\|_X}(c, d).$$

By the assumption of the lemma, $S\|_X = T\|_X$, and together with $X \subseteq L$, we have

$$\begin{aligned} \text{lca}_{S\|_X}(a, b) = \text{lca}_{S\|_X}(c, d) &\iff \text{lca}_{T\|_X}(a, b) = \text{lca}_{T\|_X}(c, d) \\ &\iff \text{lca}_{T\|_L}(a, b) = \text{lca}_{T\|_L}(c, d). \end{aligned}$$

Case 2: u is on the spine, i.e., $u = p(S_\ell)$ for some $1 \leq \ell \leq p$. Since the ancestor of a and b is $p(S_\ell)$, one of the leaves, say a , must be at S_ℓ , and the other, b , must be at some sidetree S_h below S_ℓ . Similarly, one of c, d , say c is at S_ℓ and d is at some sidetree S_v below S_ℓ . Below, we show that a, c must be in the same sidetree $T_{\ell'}$ in T , and b and d must be in some sidetrees below $T_{\ell'}$. Then,

$$\text{lca}_{T\|_L}(a, b) = p(T_{\ell'}) = \text{lca}_{T\|_L}(c, d).$$

We first prove that a and c are in the same sidetree of T . Since a, c are at S_ℓ , $\{a, c\} \subseteq X$ where X is either A , B , or C (as in Case 1). If $X = B$, then $\{a, c\} \subseteq B \subseteq \mathcal{L}(T_j)$, and the two leaves are at the same sidetree of T . Now, suppose $X = A$ (the case when $X = C$ is similarly). Note that $\kappa \in K \cap \mathcal{L}(S_{\text{top}}) \subseteq A$ and $\text{lca}_S(a, \kappa) = \text{lca}_S(c, \kappa) (= p(S_\ell))$. Then, by $S\|_A = T\|_A$, we conclude $\text{lca}_T(a, \kappa) = \text{lca}_T(c, \kappa)$. Note that this is possible only when the leaves a, c are at the same sidetree $T_{\ell'}$ of T .

Now, we show that b is at some sidetrees below $T_{\ell'}$, the sidetree containing a and b . We can handle d similarly. Recall that a is at S_ℓ and b is at S_h where $\ell < h$. Note that if a, b are in different sets of A , B , and C , say $a \in A$ and $b \in B$, then $a \in \mathcal{L}(T_1, T_2, \dots, T_{j-1})$ and $b \in \mathcal{L}(T_j)$, and thus b is at some sidetree below $T_{\ell'}$. Otherwise, $\{a, b\} \subseteq X$ where X can either be A or C (they are not be in B because a, b are at different sidetrees). Note that $\kappa \in X$ and $\text{lca}_{S\|_X}(a, \kappa) = \text{lca}_{S\|_X}(a, b) (= p(S_\ell))$. Then, by $S\|_X = T\|_X$, we conclude $\text{lca}_{T\|_X}(a, \kappa) = \text{lca}_{T\|_X}(a, b)$, which is $p(T_{\ell'})$. This is possible only when b is at some sidetree below $T_{\ell'}$.

To summarize, we have proved in both cases that $\text{lca}_{S\|_L}(a, b) = \text{lca}_{S\|_L}(c, d) \implies \text{lca}_{T\|_L}(a, b) = \text{lca}_{T\|_L}(c, d)$. The other direction \impliedby can be proved similarly. Then, by Fact 1, we conclude $S\|_L = T\|_L$ and the lemma is proved.

To show that L is the largest label set in $\text{CAST}(S, T, K)$. we need to prove two nice structural properties on every label set in $\text{CAST}(S, T, K)$.

Lemma 3. $H \in \text{CAST}(S, T, K)$

$$\begin{aligned} H \cap \mathcal{L}(S_i) &= H \cap \mathcal{L}(T_j) \\ H \cap \mathcal{L}(S_{\text{top}}) &= H \cap \mathcal{L}(T_{\text{top}}) \\ H \cap \mathcal{L}(S_{\text{bot}}) &= H \cap \mathcal{L}(T_{\text{bot}}) \end{aligned}$$

Since $H \in \text{CAST}(S, T, K)$, we have $K \subseteq H$ and $S||_H = T||_H$. We first prove (1) by contradiction. Suppose $H \cap \mathcal{L}(S_i) \neq H \cap \mathcal{L}(T_j)$. Recall that S_i and T_j are the highest side trees that contain some label in K . Thus, $K \cap \mathcal{L}(S_i) \neq \emptyset$ and $K \cap \mathcal{L}(T_j) \neq \emptyset$. We consider two cases.

Case 1: $K \cap \mathcal{L}(S_i) = K \cap \mathcal{L}(T_j)$. Then, there is a label $\tau \in K$ that is in both S_i and T_j . Since we assume $H \cap \mathcal{L}(S_i) \neq H \cap \mathcal{L}(T_j)$, there is a label a that is in, say $H \cap \mathcal{L}(S_i)$, but not in $H \cap \mathcal{L}(T_j)$. It follows that a is in S_i , and is in some tree T_ℓ where $\ell \neq j$. Note that

$$\text{lca}_S(\tau, \kappa) = \text{lca}_S(a, \kappa) = p(S_i),$$

but

$$\text{lca}_T(\tau, \kappa) = p(T_j) \neq p(T_\ell) = \text{lca}_T(a, \kappa).$$

Furthermore, $\{a, \tau, \kappa\} \subseteq H$, and thus the condition required by Fact 1 can never be satisfied by $S||_H$ and $T||_H$. Hence, the two trees cannot be leaf-label isomorphic and hence $H \notin \text{CAST}(S, T, K)$; a contradiction.

Case 2: $K \cap \mathcal{L}(S_i) \neq K \cap \mathcal{L}(T_j)$. Then, there is a label a in, say, $K \cap \mathcal{L}(S_i)$, but is not in $K \cap \mathcal{L}(T_j)$. It follows that a is in S_i , and is in some tree T_ℓ . Note that $\ell \geq j$ because by construction of the algorithm, T_1, T_2, \dots, T_{j-1} do not contain any label in K . Let τ be a label in $K \cap \mathcal{L}(T_j)$. Suppose τ is at some sidetree S_h of S . Again, $h \geq i$ because S_1, S_2, \dots, S_{i-1} do not contain any label in K . Note that no matter whether $h = i$ or $h > i$,

$$\text{lca}_S(\tau, \kappa) \neq \text{lca}_S(\tau, a)$$

but

$$\text{lca}_T(\tau, \kappa) = \text{lca}_T(\tau, a) = p(T_j).$$

Similar to Case 1, we conclude $H \notin \text{CAST}(S, T, K)$; another contradiction. Hence, the only possible conclusion is $H \cap \mathcal{L}(S_i) = H \cap \mathcal{L}(T_j)$, and (1) is proved.

Now, we prove (2) by contradiction ((3) can be proved similarly). Assume that $H \cap \mathcal{L}(S_{\text{top}}) \neq H \cap \mathcal{L}(T_{\text{top}})$. Then there is a label a that is in, say, $H \cap \mathcal{L}(S_{\text{top}})$, but not in $H \cap \mathcal{L}(T_{\text{top}})$. Thus, $a \in S_\ell$ for some $\ell < i$, and $a \in T_h$ for some $h \geq j$. Since we have proved (1), we conclude there is a label $\tau \in H$ that are in both S_i and T_j . Note that,

$$\text{lca}_S(a, \kappa) = \text{lca}_S(a, \tau) = p(S_\ell),$$

and no matter whether $h = j$ or $h > j$,

$$\text{lca}_T(a, \kappa) \neq \text{lca}_T(a, \tau).$$

Since $\{a, \tau, \kappa\} \subseteq H$, $S||_H \neq T||_H$ and $H \notin \text{CAST}(S, T, K)$; a contradiction.

Lemma 4. $H \in \text{CAST}(S, T, K)$ $X = H \cap \mathcal{L}(S_{\text{top}})$, $Y = H \cap \mathcal{L}(S_i)$ $Z = H \cap \mathcal{L}(S_{\text{bot}})$

$$X \in \text{CAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}})).$$

$$\begin{aligned} Y &\in \text{CAST}(S_i, T_j, K \cap \mathcal{L}(S_j)), \\ Z &\in \text{CAST}(S_{\text{bot}}, T_{\text{bot}}, K \cap \mathcal{L}(S_{\text{bot}})) \end{aligned}$$

Since $H \in \text{CAST}(S, T, K)$, we have $K \subseteq H$ and $S\|_H = T\|_H$. To prove (1), it suffices to show that (i) $K \cap \mathcal{L}(S_{\text{top}}) \subseteq X$ and (ii) $S_{\text{top}}\|_X = T_{\text{top}}\|_X$. Note that (i) is trivial because $K \subseteq H$ implies $K \cap \mathcal{L}(S_{\text{top}}) \subseteq H \cap \mathcal{L}(S_{\text{top}}) = X$. To show (ii), we observe that

$$S_{\text{top}}\|_X = (S\|_{\mathcal{L}(S_1, S_2, \dots, S_{i-1}) \cup \{\kappa\}})\|_X = S\|_X \quad (1)$$

because $S_{\text{top}} = S\|_{\mathcal{L}(S_1, S_2, \dots, S_{i-1}) \cup \{\kappa\}}$ and $X \subseteq \mathcal{L}(S_{\text{top}}) = \mathcal{L}(S_1, S_2, \dots, S_{i-1}) \cup \{\kappa\}$.

Furthermore,

$$T_{\text{top}}\|_X = (T\|_{\mathcal{L}(T_1, T_2, \dots, T_{j-1}) \cup \{\kappa\}})\|_X = T\|_X \quad (2)$$

because $T_{\text{top}} = T\|_{\mathcal{L}(T_1, T_2, \dots, T_{j-1}) \cup \{\kappa\}}$ and $X = H \cap \mathcal{L}(S_{\text{top}}) = H \cap \mathcal{L}(T_{\text{top}})$ (Lemma 3), which is a subset of $\mathcal{L}(T_1, T_2, \dots, T_{j-1}) \cup \{\kappa\}$. Since $S\|_H = T\|_H$, we have

$$S\|_X = (S\|_H)\|_X = (T\|_H)\|_X = T\|_X. \quad (3)$$

Combining (1),(2), and (3), we conclude $S_{\text{top}}\|_X = T_{\text{top}}\|_X$.

We prove (2) and (3) similarly.

Now, we are ready to prove that the label set L returned by `ConstMCAST` is indeed in $\text{MCAST}(S, T, K)$, and hence the procedure `ConstMCAST` is correct.

Theorem 3. $L \in \text{MCAST}(S, T, K)$ if L is returned by `ConstMCAST`.

We prove the theorem by induction on the size of K . If $|K| \leq 1$, then L is returned by `SimpleMCAST`(S, T, K) correctly.

Suppose $|K| > 1$. Then L is the union of A, B and C , which are returned recursively by `ConstMCAST`($S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}})$), `ConstMCAST`($S_i, T_j, K \cap \mathcal{L}(S_i)$), and `ConstMCAST`($S_{\text{bot}}, T_{\text{bot}}, K \cap \mathcal{L}(S_{\text{bot}})$), respectively. Note that the size of $K \cap \mathcal{L}(S_{\text{top}})$, $K \cap \mathcal{L}(S_i)$, and $K \cap \mathcal{L}(S_{\text{bot}})$ are strictly smaller than $|K|$. By induction, we have

1. $A \in \text{MCAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}}))$,
2. $B \in \text{MCAST}(S_i, T_j, K \cap \mathcal{L}(S_i))$, and
3. $C \in \text{MCAST}(S_{\text{bot}}, T_{\text{bot}}, K \cap \mathcal{L}(S_{\text{bot}}))$,

and Lemma 2 asserts that $L \in \text{CAST}(S, T, K)$.

To show that $L \in \text{MCAST}(S, T, K)$, consider any label set $H \in \text{MCAST}(S, T, K)$. Let $X = H \cap \mathcal{L}(S_{\text{top}})$, $Y = H \cap \mathcal{L}(S_i)$, and $Z = H \cap \mathcal{L}(S_{\text{bot}})$. By Lemma 4, we have $X \in \text{CAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}}))$, $Y \in \text{CAST}(S_i, T_j, K \cap \mathcal{L}(S_i))$ and $Z \in \text{CAST}(S_{\text{bot}}, T_{\text{bot}}, K \cap \mathcal{L}(S_{\text{bot}}))$. Recall that $A \in \text{MCAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}}))$ and this implies $|A| \geq |X|$. Similarly, $|B| \geq |Y|$ and $|C| \geq |Z|$. It follows that

$$|L| = |A| + |B| + |C| \geq |X| + |Y| + |Z| = |H|.$$

Consequently, L is no smaller than any label set in $\text{CAST}(S, T, K)$, and hence $L \in \text{MCAST}(S, T, K)$.

We now estimate the running time of $\text{ConstMCAST}(S, T, K)$. From the design of the procedure, it can be verified that $|K \cap \mathcal{L}(S_{\text{top}})| = 1$. Hence, the first recursive call $\text{ConstMCAST}(S_{\text{top}}, T_{\text{top}}, K \cap \mathcal{L}(S_{\text{top}}))$ invokes immediately the procedure SimpleMCAST , which will stop in $O(n_1 \log n_1)$ time where $n_1 = |S_{\text{top}}| + |T_{\text{top}}|$. In fact, it is easy to see that the execution of $\text{ConstMCAST}(S, T, K)$ is composed of a sequence of calls on $\text{SimpleMCAST}(U_i, V_i, K_i)$ and thus the total running time is $O(\sum n_i \log n_i)$ where $n_i = |U_i| + |V_i|$. Observe that for any label that is not in K , the label will be in exactly one U_i and one V_i . Furthermore, every U_i and every V_i have at most one label in K . It follows that $\sum n_i = O(n)$. Hence, we have the following theorem.

Theorem 4. $\text{ConstMCAST}(S, T, K) \dots \dots O(\sum n_i \log n_i) = O(n \log n) \dots$

5 Concluding Remarks

In this paper, we introduce the maximum constrained agreement subtree problem and give an $O(n \log n)$ time algorithm for solving the problem when the two given input trees are binary. The time complexity of our algorithm matches the fastest algorithm for constructing classical maximum agreement subtree of two binary algorithms. There are many interesting problems related to MCAST. We list below a few of them.

1. Design efficient algorithm for constructing MCAST when the two given input trees have degrees bounded by some constant d .
2. Design efficient algorithm for constructing MCAST when there is no constraint on the degree of the input tree.
3. Design efficient algorithm for constructing MCAST for unrooted evolutionary tree.

References

1. R. Cole, M. Farach, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.
2. M. Farach and M. Throup. Optimal evolutionary tree comparison by sparse dynamic programming. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 770–779, 1994.
3. M. Farach and M. Throup. Fast comparison of evolutionary trees. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 481–488, 1995.
4. C. Finden and A. Gordan. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.
5. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

6. S. Kannan, E. L. Lawler, and T. Warnow. Determining the evolutionary tree. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 475–484, 1990.
7. M.Y. Kao. Tree contractions and evolutionary trees. *SIAM Journal on Computing*, 27:1592–1616, 1998.
8. M.Y. Kao, T.W. Lam, W.K. Sung, and H.F. Ting. A decomposition theorem for maximum weight bipartite matchings with applications in evolution trees. In *Proceedings of the 7th Annual European Symposium on Algorithms*, pages 438–449, 1999.
9. M.Y. Kao, T.W. Lam, W.K. Sung, and H.F. Ting. An even faster and more unifying algorithm comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 20(2):212–233, 2001.
10. D. Keselman and A. Amir. Maximum agreement subtree in a set of evolutionary trees— metrics and efficient algorithms. In *Proceedings of 35th Annual Symposium on the Foundations of Computer Sciences*, pages 758–769, 1994.
11. E. Kubicka, G. Kubicki, and F. McMorris. An algorithm to find agreement subtrees. *Journal of Classification*, 12:91–99, 1995.
12. M. Steel and T. Warnow. Kaikoura tree theorems: computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1994.

Planning the Transportation of Multiple Commodities in Bidirectional Pipeline Networks^{*}

Artur Alves Pessoa

PUC-Rio, Informatics Department, Rua Marquês de São Vicente 225, RDC,
4^o andar, CEP 22453-900, Rio de Janeiro - RJ, Brazil
artur@inf.puc-rio.br

Abstract. PTP is a combinatorial model for oil pipeline transportation. It uses a directed graph G and a set of product *batches*, where each batch has a corresponding destination node. The graph G has an associated initial state where each arc contains a non-empty sequence of batches and each node contains a set of batches. A valid movement is to *shift* the batches of a given arc $a = (i, j)$ in such a way that one batch from the node i is inserted in the beginning of the arc a and another batch, removed from the end of the arc a , is inserted in j . The goal is to reach a state where all batches from a given subset are stored at their destinations. For the general PTP, deciding whether or not a feasible plan exists is proved to be \mathcal{NP} -hard.

In this paper, we introduce the CBPTP, a new variation of PTP. In this variation, arc contents may be shifted in both directions. Moreover, we generalized the batch destination nodes by assigning a commodity to each batch and defining node demands. For the general CBPTP, we give a polynomial algorithm for constructing feasible plans for the CBPTP, when they exist.

1 Introduction

Pipelines play an important role in the transportation of petroleum and its derivatives, since it is the most effective way to transport large volumes over large distances. However, planning the transportation of petroleum products through a relatively small pipeline network may be a very hard task [2, 4, 6, 7, 3]. This scenario makes transportation through oil pipeline networks a problem of high relevance.

The pipeline transportation problem has an unique characteristic, which distinguishes it from other transportation methods: it uses stationary carriers whose (liquid) cargo moves rather than the more usual moving carriers of stationary cargo. Typically, each oil pipeline is a few inches wide and several miles long.

^{*} Partially supported by FAPERJ (Proc. E-26/150.715/2003) and by CNPQ, through Edital Universal 01/2002 (Proc. 476817/2003-0).

As a result, reasonable amounts of distinct products can be transported through the same pipe with a very small loss due to mixing at liquid boundaries. Another important characteristic of oil pipelines is that they must always, for safety reasons, be pressurized. That is, each pipeline must be completely full of liquid. Hence, assuming incompressible fluids, an elementary pipeline operation is the following: pump an amount of product into the pipeline and remove the same amount of product from the opposite side. Observe that the product inserted in one side is not necessarily the same as the one removed from the other side, as shown in Figure 1.(a). This figure shows the content of an oil pipeline that is filled by the two (liquid) products B and A, from left to right. In this case, to insert an amount of the product C in the left side, one must remove the same amount of A from the right side. As a result, product B is displaced to the right, inside the oil pipeline. Figure 1.(b) shows the content of the same pipeline after this operation.

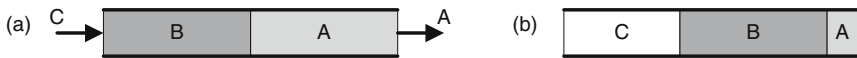


Fig. 1. (a) the content of an oil pipeline; (b) the content of the same pipeline after inserting an amount of product C

Throughout this paper, we use the term batch to denote a contiguous amount of some product in a pipeline network. This term is commonly used in the context of oil pipelines. Further information about oil pipelines can be found in [8].

1.1 Previous Work

The idea to study simplified combinatorial models for the pipeline transportation problem was introduced by Hane and Ratliff [2]. The authors propose a model under the following assumptions: demands cyclically repeat over the time, the network graph is a directed tree, batches have given destination nodes and no due dates.

The Pipeline Transportation Problem (PTP) is proposed in [5, 6] for general graphs and non-cyclic orders. PTP models a pipeline system through a directed graph G , where each node represents a location and each directed arc represents a pipe, with a corresponding flow direction. As in Hane’s model, the flow inside each pipe is assumed to be unidirectional. PTP also defines a set of batches with given destination nodes. The graph G has an associated initial state where each arc contains a non-empty sequence of batches and each node contains a (possibly empty) set of batches. All batches have unitary volumes. Since pipes must always be full, some batches must be used to fill them at the end of the transportation. These batches are not delivered. Due to this fact, PTP defines a subset of batches, called *delivered* batches, that are not necessarily delivered at the end of a feasible pumping sequence. In this case, a feasible solution to PTP is a pumping sequence that delivers all *delivered* batches.

In [6], the problem of finding a feasible solution to PTP is proved to be \mathcal{NP} -hard, even if G is acyclic. Moreover, the authors introduce the Synchronous PTP (SPTP), a special case of PTP where all further batches are initially stored at nodes. The problem of finding a minimum pumping cost solution to SPTP is called SPTOP. In the same work, the authors also introduce the BPA algorithm, that finds feasible solutions to SPTP in polynomial time. If G is acyclic, then these solutions are also optimal for the SPTOP. The authors also analyze the complexity of finding minimum makespan solution to the SPTP in [7]. This problem is called SPTMP. They prove that, for any fixed $\epsilon > 0$, there is no $\eta^{1-\epsilon}$ -approximate algorithm for SPTMP unless $\mathcal{P} = \mathcal{NP}$, where η is the input size. This result also holds if the graph G is both planar and acyclic.

Recently, another combinatorial model based on oil pipelines, called Pipesworld, was proposed for a solver competition [3]. The authors also describe many interesting examples of difficulties introduced by the Pipesworld model. We point out that the Pipesworld model can be viewed as a variation of the PTP model where pipe contents can flow in both directions, nodes have capacity constraints and consecutive batches in pipes have compatibility restrictions.

1.2 The Problem

In this paper, we address a variation of PTP for planning the transportation of multiple commodities in bidirectional pipeline networks. We refer to our model as the Commodity Bidirectional PTP (CBPTP). As in the Pipesworld model, CBPTP allows pipe contents to flow in both directions. On the other hand, product batches have no capacity constraints or compatibility restrictions. Moreover, instead of assigning destination nodes to the batches, we use a more general model. Each batch has an associated commodity, and each node i in G has an associated demand $d(i, c)$ for each commodity c . In this case, a feasible solution to the CBPTP is a pumping sequence that delivers at least $d(i, c)$ batches of the commodity c at the node i , for every node i and commodity c . Hence, the model must also determine a destination node for each batch. Observe that we may model the case where the batches' destination nodes are given as a special case of the CBPTP by assigning a different commodity to each batch. Let us refer to this last case as the Assigned Bidirectional PTP (ABPTP).

We point out that, since all pipes can flow in both directions, the arc directions are merely a convention to represent the pipe contents. Hence, we define an undirected graph \bar{G} as the graph obtained from G by replacing all its directed arcs with the corresponding (undirected) edges. For the sake of simplicity, let us assume that G has no parallel arcs (even with different directions).

1.3 Results

For the general CBPTP, we reduce the instance feasibility to the problem of finding a perfect matching in a bipartite graph. Moreover, we give a polynomial algorithm for constructing feasible plans for the CBPTP, when they exist.

Our approach to solve the CBPTP is the following. First, we obtain necessary and sufficient feasibility conditions for the ABPTP. Later, we observe that, given a feasible instance I of the CBPTP, we can use the previous conditions to assign destination nodes to the batches of I so that the resulting instance of the ABPTP is feasible. For that, we show that such assignment must correspond to a perfect matching in a bipartite graph.

To solve the ABPTP, we start with a solution for the special case where the graph G has a single arc. This solution is generalized to . . . the bridges of the graph \tilde{G} . By handling a bridge e , we mean moving to a state where every non-further batch can reach its destination node without passing by the pipe that corresponds to e . As a result, handling a bridge involves placing all batches that must cross the corresponding pipe at the correct side. Then, we use a decomposition of the graph \tilde{G} into bridges and 2-edge-connected components [1]. Whenever the current graph \tilde{G} has at least one bridge, we solve the problem as follows: choose a bridge e on \tilde{G} ; handle e , and recursively solve the subproblems that correspond to the two subgraphs of \tilde{G} connected by e . Here, we assume that the graph \tilde{G} is initially connected. For the 2-edge-connected components of \tilde{G} , we assign an unique flow direction for each corresponding pipe so that the resulting directed graph is strongly connected. After that, we solve the corresponding subproblem through a generalization of the approach introduced in [6].

We remark that bridges are usually found in the Brazilian oil pipeline networks, which are our motivating examples. For example, the clean oil network in São Paulo State has two long bridges.

1.4 Paper Organization

This paper is organized as follows. In Section 2, we formalize the CBPTP model and introduce some additional notations. In Section 3, we describe our polynomial time planning algorithm. In the last section, we present our final remarks.

2 The CBPTP Model

In this section, we describe the both the CBPTP and the ABPTP models. First, we describe the pipeline system, batches, pipe contents, and allowed operations for the CBPTP. Then, we introduced the differences between the CBPTP and the ABPTP.

Pipeline System

Let $G = (N, A)$ be a directed graph, where N is the set of n nodes and A is the set of m arcs. Given an arc $a = (i, j) \in A$, we say that i is the start node of a and j is the end node of a . Arcs represent pipes and nodes represent locations. Each arc $a \in A$ has an associated integer capacity $v(a)$. Moreover, we divide each arc a into $v(a)$ pipe positions. We also define the set of all pipe positions $A' = \{(a, l) | a \in A \text{ and } l \in \{1, \dots, v(a)\}\}$.

Batches

Let L be a set of r unitary volume batches, and C a set of commodities. Each $b \in L$ has an associated commodity $c(b)$. Moreover, each node $i \in N$ has an associated integer demand $d(i, c)$ for each commodity $c \in C$.

Pipe Contents

Pumping a batch into a pipe requires a non negligible amount of time. However, we only consider the instants where each arc $a \in A$ contains exactly $v(a)$ integral batches. As a result, any solution to this model generates a discrete sequence of states, where the positions of all batches are well-defined.

Let us use $p_t(b)$ to denote the position of batch b at state t . If $p_t(b) = (a, l) \in A'$, then batch b is located at the l th position of arc a at state t . Otherwise, if $p_t(b) = i \in N$, then batch b is stored at node i . Furthermore, the content of a given arc a at a given state t is represented by a list of batches $[b_1, b_2, \dots, b_{v(a)}]$, where the l th batch in this list is contained at the pipe position (a, l) , for $l = 1, 2, \dots, v(a)$.

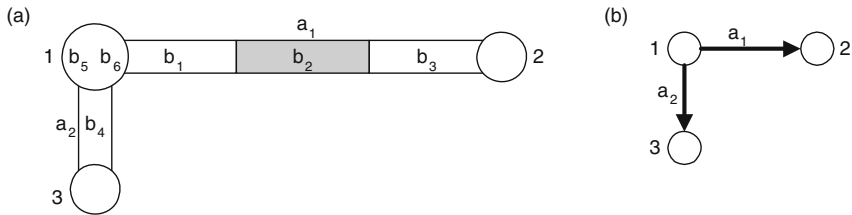


Fig. 2. (a) The contents of a pipeline system; (b) the corresponding graph

As an example, Figure 2.(a) represents the pipe contents corresponding to the graph of Figure 2.(b). Observe that the system has two pipes $a_1 = (1, 2)$ and $a_2 = (1, 3)$. The capacities of a_1 and a_2 are $v(a_1) = 3$ and $v(a_2) = 1$, respectively. Let us assume that Figure 2.(a) corresponds to state t . In this case, we have $p_t(b_1) = (a_1, 1)$, $p_t(b_2) = (a_1, 2)$, $p_t(b_3) = (a_1, 3)$, and $p_t(b_4) = (a_2, 1)$, since the contents of a_1 and a_2 are respectively represented by the lists $[b_1, b_2, b_3]$ and $[b_4]$. Furthermore, we have $p_t(b_5) = p_t(b_6) = 1$ since both b_5 and b_6 are stored at node 1.

At the initial state (state 0), the position $p_0(b)$ of each batch b is given.

Operations

A solution for the model is a list Q of elementary pipeline operations (EPO), defined as follows. Let $a = (i, j)$ be an arc of G , whose content at a given state t is given by the list $[b_1, b_2, \dots, b_{v(a)}]$. Moreover, let b be a batch stored either at node i or at node j , in this moment. An EPO is a pair (b, a) , denoting that b is pumped into a . Suppose that (b, a) is executed after the state t . If b was previously stored at node i , then we refer to this EPO as a . . . EPO.

Otherwise, if b was previously stored at node j , then we refer to this EPO as a *reverse* EPO. In the case of a direct EPO, the contents of a at state $t + 1$ are given by the list $[b, b_1, b_2, \dots, b_{v(a)-1}]$ and $b_{v(a)}$ is stored at the node j . If a reverse EPO was performed, then the contents of a at state $t + 1$ are given by the list $[b_2, b_3, \dots, b_{v(a)}, b]$ and b_1 is stored at the node i . Let q be the size of a given sequence Q of EPO's. We say that Q is feasible when, for every node $i \in N$ and every commodity $c \in C$, we have at least $d(i, c)$ batches associated to the commodity c , stored at node i when the state is q .

The ABPTP Model

For the ABPTP, each batch b has a fixed destination node $f(b) \in N \cup \{\phi\}$. Each batch b with $f(b) = \phi$ is called a *phantom* batch. These batches have no destination node assigned and may be used to fill the pipes at the end of the plan. All other batches are called *real* batches. Also, we say that Q is feasible for the ABPTP when every batch b with $f(b) \neq \phi$ is stored at the node $f(b)$, when the state is q .

3 Planning Algorithm

In this section, we describe a polynomial algorithm that solves the CBPTP. Given an instance I of the CBPTP, our algorithm either finds a feasible solution I or determines that I is infeasible. Let us refer to this algorithm as the Bridge algorithm, which is motivated by the fact that handling the bridges of the graph \tilde{G} is the most difficult part.

As mentioned in the introduction, we assign an unique flow direction for each pipe of a 2-edge-connected component of \tilde{G} so that the resulting directed subgraph is strongly connected. In [9], it is shown that such directions can be assigned for any 2-edge-connected undirected graph. In fact, these directions can be obtained in a linear time through a Depth First Search. For the sake of simplicity, let us assume without loss of generality that the assigned flow directions coincide with arc orientations of G . Moreover, let us refer to any arc of G that corresponds to bridge in \tilde{G} as a bridge also.

3.1 Network Excess

Now, we introduce the concept of excess, which is extensively used throughout this section.

Definition 1. Let $G = (N, A)$ be a graph and $G' = (N', A')$ be a subgraph of G . Let t be a state of G' . The excess of G' at state t is defined as $x_t(G') = |\{b \in L | p_t(b) \in N'\}|$

Observe that no EPO can change the excess of G . As a result, this excess is constant over the states generated by any plan. Hence, we define $X = x_0(G) = x_1(G) = \dots = x_q(G)$, for any plan Q of size q .

Throughout this paper, we may also use the term excess to refer to a set of batches stored at a node. In this case, given an arc $a = (i, j)$, if k batches

are pumped from the node i into a , then we may say that an excess of size k is moved from i to j . Note that the batches received by j may be completely different from the batches pumped from i . Observe also that we can move an excess from j to i , executing reverse EPO's on a .

Now, let us consider a path in G (possibly containing both direct and reverse arcs) that starts at a node i_0 and visits the nodes i_1, i_2, \dots, i_l in this order. In this case, an excess of size k may be moved from i_0 to i_l as follows: move it from i_0 to i_1 , then move it from i_1 to i_2 and so on until i_l receives an excess of size k . Observe that, since G is connected, any excess can be moved from any node to any node in G .

3.2 Feasibility Conditions for the ABPTP

We have already mentioned that we solve the CBPTP through a reduction to the ABPTP. Here, we introduce a necessary and sufficient set of conditions for the feasibility of the ABPTP. Although the necessity is easy to prove, the sufficiency is only obtained through a planning algorithm. Hence, we start stating these conditions only as necessary conditions. For that, let us introduce some additional definitions.

Given a bridge $a = (i, j)$ of G , let us define $S(a, G)$ and $S'(a, G)$ as the two subgraphs of G connected by a , where i and j are nodes of $S(a, G)$ and $S'(a, G)$, respectively. In this case, we have the following definitions.

Definition 2.

$$y_t(b, a) = \begin{cases} l & \dots p_t(b) = (a, l); \\ 0 & \dots p_t(b) \dots \dots S(a, G); \\ v(a) + 1 & \dots p_t(b) \dots \dots S'(a, G). \end{cases}$$

Definition 3.

$$t = 0, 1, \dots, q. \quad w_t(a) = x_t(S'(a, G)) \quad w'_t(a) = v(a) - x_t(S(a, G)) + 1$$

Next, we show that, if $v(a) \geq X$, then $w'_t(a)$ ($w_t(a)$) is the minimum (maximum) position of a batch b w.r.t. a , when the state is t , such that b can be delivered at a node of $S'(a, G)$ ($S(a, G)$). As we show later, any batch can cross any bridge a such that $v(a) < X$. On the other hand, the next lemma shows that this is not true for bridges with $v(a) \geq X$.

Lemma 1.

$$v(a) \geq X \quad y_0(b, a) < w'_0(a) \quad (y_0(b, a) > w_0(a)), \dots, \dots, \dots b \dots \dots S'(a, G) \quad (S(a, G))$$

Let us defer it to the extended version of this paper. ■

Observe that Lemma 1 leads to the following necessary feasibility condition associated to each non-further batch $b \in L$: for every bridge a of G , if $y_0(b, a) < w'_0(a)$ ($y_0(b, a) > w_0(a)$) then $f(b)$ must be a node of $S(a, G)$ ($S'(a, G)$). Moreover, since we must have enough further batches to fill all the pipes, we have the following fact.

Fact 1.

Now, we are ready to state the following theorem. Recall that A' is the set of all pipe positions.

Theorem 1.

- (.) $\forall b \in L \exists a \in G, y_0(b, a) < w'_0(a)$
- (.) $y_0(b, a) > w_0(a) \implies f(b) \in S(a, G) \cup S'(a, G)$
- (.) $|A'| \leq |\{b \in L | f(b) = \phi\}|$

The necessity to satisfy these conditions is a consequence of both Lemma 1 and Fact 1. On the other hand, the sufficiency is only proved by the algorithm described throughout this section. ■

3.3 Reduction from CBPTP to ABPTP

Here, we show how to use a polynomial algorithm for the ABPTP to solve the CBPTP in polynomial time. Clearly, we can transform any instance I of the CBPTP in an instance I' of the ABPTP by assigning destination nodes to all batches of I so that all demands of I are satisfied. However, we only have a reduction if such an assignment maintains the feasibility of I , that is, every feasible instance I leads to a feasible instance I' .

Observe that the conditions of Theorem 1 gives a set of possible destinations for each batch which is independent from the destinations assigned to the other batches. The only restriction is that the given demands and required further batches are satisfied. As a result, the feasibility problem can be modeled as an matching problem in a bipartite graph, where batches are nodes from one partition and destinations are nodes from the other. We leave the construction of this graph to an extended version of this paper.

3.4 One-Pipe Networks

Here, describe our procedure to handle a bridge $a = (i, j)$ of G , assuming that a is the only arc of G . Let us refer to this procedure as the One-pipe Procedure. Later, in Subsection 3.6, we show how to generalize our solution to any network graph. As we previously define, to handle a bridge a , we must move the pipeline network to a state where every non-further batch b such that $f(b)$ is a node of $S(a, G) \cup S'(a, G)$ is already stored in this subgraph. In our case, the two subgraphs are single nodes.

We divide in two cases: (i) $v(a) \geq X$; (ii) $v(a) < X$.

In the case (i), a can be handled in four steps:

One-pipe Procedure – Case (i)
Step 1: Pump all batches from j into a .
Step 2: Pump all further batches from i into a .
Step 3: Pump all non-further batches from i into a .
Step 4: Pump all further batches from j into a .

Observe that, if there are enough further batches to fill a , then a is filled with only further batches after the Step 4. Since, by Lemma 1, no batch can cross a , we conclude also that all non-further batches are properly delivered after this step (if the instance is feasible).

Now, let us consider the case (ii). The crucial observation is that, if all excess of G is stored at the node i , then the first batch b pumped from i into a can reach the node j . This is true because the excess of G minus one is sufficient to fill a . On the other hand, we may have to pump all the batches of i into a to get b stored at j . Analogously, if all excess of G is stored at the node j , then the first batch pumped from j into a can reach the node i . This observations suggest an yo-yo like procedure to handle bridges. This procedure performs the following steps:

One-pipe Procedure – Case (ii)
Step 1: Pump all batches from j into a .
Step 2: Pump into a every non-further batch b from i such that $f(b) = j$.
Step 3: Pump all further batches from i into a .
Step 4: Pump all remaining non-further batches from i into a .
Step 5: Pump into a every non-further batch b from j such that $f(b) = i$.
Step 6: Pump all further batches from j into a .
Step 7: If there is a non-further batch b not stored at $f(b)$ then go to Step 1.

Observe that, the non-further batches stored at the corresponding destination nodes are the last pumped into a . As a result, since we must have enough further batches to fill a , every non-further batch b that is stored at $f(b)$ when executing the Step 7 returns to $f(b)$ before the next execution of this step. Moreover, due to the previous observations, we have that the number of non-further batches stored at the corresponding destination nodes increases on each execution of the Step 7. Hence, the condition of Step 7 is satisfied after at most r iterations, where r is the total number of batches in the network.

3.5 Selecting Batches from Subgraphs

Let $a = (i, j)$ be a bridge of G . If a is the only pipe of G , then we can select any batch from i or j and pump it into a . Here, we describe a method to select a batch b either from $S(a, G)$ or from $S'(a, G)$ and pump it into a , where G represents any pipeline network. Let us refer to this method as the Selection Procedure.

The Selection Procedure uses the Cycling Procedure, proposed in [6]. Given an arc $a' = (i', j')$ of a Strongly Connected Component (SCC) of G and a batch b stored at i' , this procedure transports b to the node j' . We point out that this procedure does not generate reverse EPO's since this is a restriction of the model used in [6]. Essentially, this procedure finds a cycle the contains a' and make the batches move along this cycle until b is stored at j' .

We shall describe the Selection Procedure only for the case where b is selected from $S(a, G)$, since the other case is analogous. This procedure uses the following three assumptions:

Assumption 1: $v(a)$ is maximum over all bridges of G ;

Assumption 2: $x_t(S(a, G)) > v(a)$, where t is the state at which b is selected;

Assumption 3: all the excess of $S(a, G)$ is stored at i ;

Later, we show how to satisfy these assumptions when solving general pipeline networks.

Next, we describe the Selection Procedure. If b is stored at i , then just pump b into a and we are done. Since, by the assumption 3, no batch in $S(a, G)$ is stored at a node other than i , in the only remaining case, b is contained in a pipe $a' = (i', j')$. In this case, we first show how to move b to a node. If a' is a bridge, then, by the assumption 1, $v(a') \leq v(a)$. In this case, it is enough to move an excess of size $v(a)$ through a' . Otherwise, if a' is not a bridge, then we start by moving an excess of size 1 to i' . In this case, we observe that the Cycling Procedure can be modified to transport b to j' . For that, we only need to remove the Step 2 (since b is already contained in a').

Now, b is already stored at a node. If this node is i , then just pump b into a and we are done. Hence, let us assume that b is stored at a node $k \neq i$. In this case, find a path \mathbf{p} in G from k to i , where the bridges (and only them) may be reverse arcs. Then, move an excess of some size from i to k so that the total excess of k increases to $v(a) + 1$ (including the batch b). After that, transport b together with an excess of size $v(a)$ through each arc a' of \mathbf{p} as follows. If a' is a bridge, then, by the assumption 1, $v(a') \leq v(a)$. In this case, it is enough to pump into a' the batch b followed by $v(a)$ batches. Otherwise, if $a' = (i', j')$ is not a bridge, then $v(a')$ may be greater than $v(a)$. Hence, we use the Cycling Procedure to transport b from i' to j' . After that, we also move an excess of size $v(a)$ from i' to j' . After transporting b through all arcs of \mathbf{p} , we have b stored at the node i . Then, we just pump b into a .

3.6 Handling Bridges

Finally, we describe the Bridge Procedure. Given an input graph G , the Bridge Procedure selects a bridge $a = (i, j)$ that maximizes $v(a)$ in G , handle it and recursively solve the two subproblems that correspond to $S(a, G)$ and $S'(a, G)$. If G has no bridge, then, by the choice of the arc directions, G is strongly connected. In this case, the SCC Procedure, that we introduce in the next subsection, is called to solve the current problem with no recursion.

Before describing the Bridge Procedure, we introduce the concept of EPO's. Let (b, a) be a direct EPO executed when the content of a is given by $[b_1, b_2, \dots, b_{v(a)}]$, for $a = (i, j)$. After this EPO, the content of a changes to $[b, b_1, b_2, \dots, b_{v(a)-1}]$, and $b_{v(a)}$ is stored at the node j . In this case, we may reverse the previous EPO by executing the reverse EPO $(b_{v(a)}, a)$, which changes the state of the network back to the state that precedes the execution of (b, a) . Observe that reverse EPO's may also be rolled back by executing the corresponding direct EPO's. We also observe that any sequence of EPO's may be rolled back. For that, we must roll back each EPO of the sequence in the reverse order of their execution.

The Bridge Procedure is a generalization of the One-pipe Procedure that uses the Selection Procedure whenever it is necessary to pump into a a batch that is not stored at a node adjacent to a . We point out that, in the generalized method, the recursive call to solve the two generated subproblems occurs while handling a , not after that. This is necessary because the recursive calls must generate subproblems whose excesses are equal to X , in order to maintain the condition (i) of Theorem 1 satisfied.

Now, let us consider the three assumptions of the Selection Procedure. The Assumption 1 is satisfied by the choice of the bridge a . The most important observation to satisfy the Assumption 2 is the following. For a given iteration of the One-pipe Procedure – Case (ii), if this iteration is not the last one, then the last $v(a)$ batches pumped from i during the Steps 2, 3, and 4 will return to the node i . This is true because, since $X > v(a)$, more than $v(a)$ batches will be pumped from j before the procedure halts. The same observation is also valid for the last $v(a)$ batches pumped from j during the Steps 5 and 6, and the Step 1 of the next iteration. As a result, these batches may be replaced by any available batches, that is, we do not need to call the Selection Procedure before executing these EPO's. This observation assures that we satisfy the assumption 2 of the Selection Procedure when generalizing the One-pipe Procedure, except for both Case (i) and the last iteration of Case (ii). To avoid calling the Selection Procedure in these exceptional cases, we recursively call the Bridge Procedure just before they occur. In this case, the subproblem that correspond to $S(a, G)$ ($S'(a, G)$) is slightly modified so that the batches that shall be pumped into a are destined to the node $i(j)$. Consequently, when the procedure returns from this recursive call, these batches are already stored at a node adjacent to a , and the Selection Procedure no longer needs to be called. Then, the original destination nodes of the batches are restored. Finally, the Assumption 3 can be easily satisfied by moving the excesses $S(a, G)$ ($S'(a, G)$) to the node $i(j)$ before calling the Selection Procedure.

Let us defer the details of this procedure for an extended version of this paper.

3.7 Strongly Connected Components

Now, we describe the SCC Procedure to solve instances of the ABPTP such that the graph G is strongly connected. As mentioned in the beginning of this section, if G has no bridge, than one can choose directions for the arcs such that G becomes strongly connected. Since these directions are merely a convention to represent the arc contents, they can be chosen without loss of generality. Recall that the SCC Procedure is called by the Bridge Procedure to solve the subproblems that correspond to the SCC's of G .

The SCC procedure is divided in two main phases: fill all arcs with further batches; and transport all non-further batches to the corresponding destination nodes.

Next, we give a pseudocode for the first phase.


```
While not all arcs in  $G$  are filled by further batches do
  Find a node  $i$  that stores a further batch;
  Find a directed path  $\mathbf{p}$  from  $i$  such that
    all arcs but the last one are filled by further batches;
  For each arc  $a$  in  $\mathbf{p}$  do
    Pump a further batch into  $a$ ;
  End-for
End-While
```

Observe that, on each iteration of the outer loop, one further batch is pumped into an arc that is not filled by further batches. Hence, we can conclude that phase 1 terminates before the number of iterations is greater than the total number of pipe positions in G .

Finally, we observe that the second phase, can be easily solved through the Cycling Procedure (see Subsection 3.5).

References

1. Shimon Even. *Graph Algorithms*. Computer Science Press, 1979.
2. Christopher A. Hane and H. Donald Ratliff. Sequencing inputs to multi-commodity pipelines. *Annals of Operations Research*, 57, 1995. Mathematics of Industrial Systems I.
3. Ruy Luiz Milidiú, Frederico Liporace, and Carlos José P. de Lucena. Pipesworld: Planning pipeline transportation of petroleum derivatives. In *Workshop on the Competition, Trento, Italy*, June 2003.
4. Ruy L. Milidiú, Eduardo S. Laber, Artur A. Pessoa, and Pablo A. Rey. Petroleum products scheduling in pipelines. In *The International Workshop on Harbour, Maritime & Industrial Logistics Modeling and Simulation*, september 1999.
5. Ruy L. Milidiú, Artur A. Pessoa, and Eduardo S. Laber. Transporting petroleum products in pipelines (abstract). In *ISMP 2000 – 17th International Symposium on Mathematical Programming*, pages 134–135, Atlanta, Georgia, USA, August 2000.
6. Ruy L. Milidiú, Artur A. Pessoa, and Eduardo S. Laber. Pipeline transportation of petroleum products with no due dates. In *Proceedings of the LATIN'2002*, pages 248–262, Canún, Mexico, april 2002.
7. Ruy L. Milidiú, Artur A. Pessoa, and Eduardo S. Laber. The complexity of makespan minimization for pipeline transportation. *Theoretical Computer Science*, 306(1-3):339–351, 2003. presented in the APPROX'2002, Rome, Italy.
8. Shell Pipeline. Shell Pipeline Company LP – About Pipelines. in the URL <http://www.shellpipeline.com>.
9. H.E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *American Mathematical Monthly*, 46:281–283, 1939.

Efficient Algorithms for the Hotlink Assignment Problem: The Worst Case Search

Artur Alves Pessoa, Eduardo Sany Laber*, and Criston de Souza

Departamento de Informática da PUC-Rio
{artur, laber, criston}@inf.puc-rio.br

Abstract. Let T be a rooted directed tree where nodes represent web pages of a web site and arcs represent hyperlinks. In this case, when a user searches for an information i , it traverses a directed path in T , from the root node to the node that contains i . In this context, hotlinks are defined as additional hyperlinks added to web pages in order to reduce the number of accessed pages per search. In this paper, we address the problem of inserting at most 1 hotlink in each web page, so as to minimize the number of accesses in a worst case search. We present a $(14/3)$ -approximate algorithm that runs in a $O(n \log m)$ time and requires a linear space, where n and m are the number of nodes (internal and external) and the number of leaves in T , respectively. We also introduce an exact dynamic programming algorithm which runs in $O(n(nm)^{2.284})$ time and uses $O(n(nm)^{1.441})$ space. By extending the techniques presented here, a polynomial time algorithm can also be obtained when $\mathcal{K} = O(1)$ hotlinks may be inserted in each page. The best known result for this problem is a polytime algorithm with constant approximation ratio for trees with bounded degree presented by Gerstel et. al. [1].

1 Introduction

Finding desired information in a large and diverse data collection as the World Wide Web is a complex task. There are two basic ways to handle information in such collections. One views the information as a non-hierarchical structure and provides a query language to extract the relevant data from the database. The other method is based on a hierarchical index to the database according to a taxonomy of categories. Examples of such indices on the Web are Yahoo (www.yahoo.com) and the Open Directory Service (www.dmoz.org).

An advantage of the first approach over the hierarchical one is that the number of human operations required to find a desired piece of information is much lower (if the right query is used). As opposed to that, in the hierarchical approach it is necessary to traverse a path in the taxonomy tree from the root to

* Partially supported by FAPERJ (Proc. E-26/150.715/2003) and by CNPQ, through Edital Universal 01/2002 (Proc. 476817/2003-0).

the desired node in the tree. Human engineering considerations further aggravate this problem since it is very hard to choose an item from a long list (a typical convenient number is 7–10 according to [1]). Thus, the degree of the taxonomy tree should be rather low and the average depth of it is therefore high. Another problem of the hierarchical approach is that the depth of an item in the taxonomy tree is not based on the access pattern. As a result, items which have very high access frequency may require long access paths each time they are needed, while items which are “unpopular” may still be very accessible in the taxonomy tree.

Hotlinks are defined as additional hyperlinks added to hierarchical structures in order to reduce the number of accessed pages per search [2]. There has been some amount of research on the problem of assigning hotlinks to web sites [3, 4, 5, 6, 7, 1, 8].

Here, we consider the version of the problem faced by the web designer in which the given site is represented by a rooted directed tree T and only the leaves contain information to be searched by the user (in the full version of this paper, we discuss how the exact algorithm proposed here can be modified to handle the more general case where internal nodes also contain information). We assume that the user always knows which link leads to the desired information. In addition, following the “obvious navigation assumption” of [6] or, equivalently, the “greedy user model” of [1], we assume that the user always follows a hotlink (u, v) from node u when searching for a leaf in the subtree rooted by v . This model reflects the reasonable case where the user has only local information concerning the structure of the site.

The previous assumption implies that we can remove some arcs in the graph obtained from T due the addition of a set of hotlinks once they will never be followed by an user. In fact, the graph obtained due to the addition of some hotlinks and the further deletion of non used arcs is a tree. As an example, the tree of Figure 1.(b) is obtained from that of Figure 1.(a) through the addition of the hotlink (u, v) and then the addition of hotlink (z, w) . Note that the arcs (x, w) and (y, v) were removed since they will never be followed by an user.

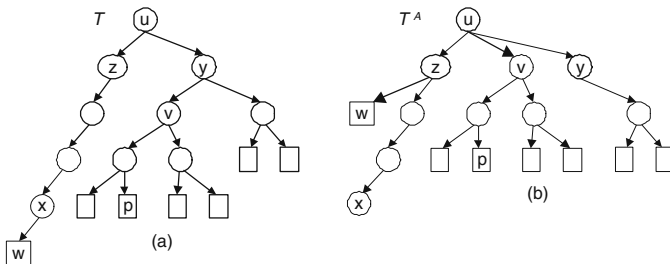


Fig. 1. (a) The tree T . (b) The improved tree T^A , where $A = \{(u, v), (z, w)\}$

1.1 Problem Definition

In order to formalize the problem, let us recall some basic definitions. The level of a node in a tree T is its distance(in number of arcs) to the root of T . We say that a node v is a descendant of another node u in T when the only path in T that connects r to v contains u . In this case, we also have that u is ancestor of v . A node u is a proper descendant (ancestor) of v if u is a descendant (ancestor) of v and $u \neq v$.

An instance of the WCHS(Worst Case Hotlink Search) problem is defined by a directed tree $T = (V, E)$, rooted at a node $r \in V$ and an integer $K \geq 1$. A solution for the WCHS problem is a hotlink assignment, defined as a set of arcs $A \subset V \times V$ ¹. A hotlink assignment A is *feasible* if and only if it satisfies the following two conditions:

- (i) for every arc $(u, v) \in A$, v is descendant of u in T ;
- (iii) for every node $u \in V$, there are at most K arcs in A that leaves u .

We say that two arcs (u, a) , (v, b) cross if u is a proper ancestor of v , v is a proper ancestor of a and a is an ancestor of b . In the “greedy user model”, we can restrict our attention to the set of feasible solutions without crossing arcs because if two arcs cross then one of them will never be followed by the user. We define a *feasible assignment without crossing arcs* as a feasible assignment without crossing arcs. For the sake of shortness whenever we use the term feasible assignment we mean feasible non-crossing assignment.

Now, we formalize the objective for the WCHS problem. For that, we define the concept of an *improved tree*.

Definition 1. Given a tree $T = (V, E)$, a hotlink assignment A and a subset $X \subseteq E$ such that $T^A = (V, (E - X) \cup A)$, we define $X = \{(u, v) \in E \mid (y, v) \in A \text{ for some } y \in V\}$

In the definition above, X is the set of arcs of E whose heads receive some hotlink from A . As an example, Figure 1.(a) shows a tree T and Figure 1.(b) shows the tree $T^{\{(u,v),(z,w)\}}$. The set X in this case is $\{(x, w), (y, v)\}$.

Given an improved tree T^A rooted at r , let $d_A(u)$ be the level of u in T^A . Moreover, let L be the subset of V that contains all leaves of T . A WC-optimal hotlink assignment to T is a feasible hotlink assignment A^* that minimizes $h(T^{A^*}) = \max_{x \in L} d_{A^*}(x)$, over all possible feasible assignments. The goal of the WCHS problem is to find a WC-optimal hotlink assignment to T . We use $h^*(T)$ to denote $h(T^{A^*})$.

Observe that some internal nodes of T may become leaves in T^A (e.g. the node x in Figure 1.(b)). By definition, these nodes do not belong to L . Hence, we refer to the nodes of L as *leaves of T* . The *depth* of a tree T' , denoted by

¹ We remark that the definition of a feasible hotlink assignment allows self loops. Although this is not necessary, it helps the description of the algorithms proposed in this paper. For practical purposes, however, we never add a self loop.

$H(T')$, is the maximum distance from the root of T' to a leaf in T' . The height takes into account all leaves including those that are not hotleaves. This measure plays an important role in the complexity of the exact algorithm proposed here. Throughout this paper, we may use H to denote the height $H(T)$ of the input tree T . In Figure 1.(b), we have $H(T^A) = 4$ while $h(T^A) = 3$.

1.2 Statement of the Results

Since we have limited space and some of the papers in this topic [3, 5] focus on the case where $\mathcal{K} = 1$, we restrict our presentation to this special case. However, by using some ideas introduced in [1] it is relatively simple to extend the exact algorithm proposed here to obtain a polynomial time algorithm for $\mathcal{K} = O(1)$. This will be discussed in the full version of this paper.

Let $n = |V|$ and $m = |L|$. We introduce an exact dynamic programming algorithm for the WCHS problem that runs in $O(n(nm)^{2.284})$ time and uses $O(n(nm)^{1.441})$ space. Furthermore, we present a $(14/3)$ -approximation greedy algorithm that runs in $O(n \log m)$ time and requires a linear space.

1.3 Related Work

The problem of assigning hotlinks to web sites was firstly studied in [3]. After that, some authors have considered the problem and its variants [4, 5, 7, 1, 8, 9]. Two user models have been considered: the “clairvoyant user model” and the “greedy user model”. In the former, the user has a map of the web site that allows him to follow a shortest path from the root to the desired information. In the latter, the user only views the current node and the arcs which leave it. Then, it always chooses the arc which allows him to get closer to the desired information (a greedy choice). Furthermore, two objective functions have been addressed: minimizing the average number of accesses for a given probability distribution on the leaves (ACHS problem) and minimizing the average number of accesses when the probability distribution is unknown. As pointed out in [1], the latter is equivalent to the problem of minimizing the number of accesses in a worst case search (the WCHS problem addressed here).

Most of the research has been conducted for the ACHS problem. When the input graph is a DAG (Directed Acyclic Graph), this problem is \mathcal{NP} -complete under both user models even for an uniform distribution on the nodes that contain information [3]. A polynomial time algorithm with constant approximation ratio for trees of bounded degree is presented in [5]. This result holds for both user models. For trees with $O(\log n)$ depth, a polynomial time algorithm under the greedy user model was independently discovered by Gerstel et. al. [1] and Pessoa et. al. [8].

The WCHS problem (unknown probability distribution nodes) was introduced in [1], where a polynomial time algorithm with constant approximation ratio for trees with bounded degree is presented. Although not mentioned in [1], the techniques presented for the ACHS problem can be easily adapted to devise

a polynomial time algorithm for the WCHS problem under the restriction that T has height $O(\log n)$.

Since we present both an exact polynomial time algorithm and a $14/3$ -approximation algorithm for general trees, our results constitute a considerable improvement over the previous known result.

1.4 Paper Organization

This paper is organized as follows. In Section 2, we introduce some additional notations and definitions. In the following section, we establish some properties of a WC-optimal assignment. These are used in the design of the algorithms proposed here. Then, in Section 4, we present the $(14/3)$ -approximation algorithm for the WCHS. Finally, in Section 5, we describe an exact polynomial algorithm for the WCHS problem.

2 Notations and Definitions

We use $T_u = (V_u, E_u)$ to denote the subgraph of T induced by the descendants of u , that is, $V_u = \{v \in V \mid v \text{ is descendant of } u\}$, and $E_u = \{(v, w) \in E \mid v, w \in V_u\}$. $T - T_u$ is used to denote the subgraph of T induced by $V - V_u$. Throughout this paper, we refer to T_u as the subtree of T rooted at u . Furthermore, we use L_u to denote the subset of V_u that contains all leaves of T_u . If $u \in L$ then we say that T_u is a leaf of T . Finally, we use S_u to denote the set of all children of node u in T .

3 Structural Properties

In this section, we prove some properties for an WC-optimal hotlink assignment. Throughout this paper, we use $\log x$ to denote $\log_2 x$.

3.1 Upper Bounds

Here, we prove upper bounds on both $h^*(T)$ and $H(T^{A^*})$. For that, we introduce the LOG algorithm for the WCHS problem, which is a quite simple recursive algorithm that starts with the tree T and during its execution it obtains a sequence of improved trees due to the addition of hotlinks. At the end, the tree T^A is obtained, where A is the set of hotlinks added by LOG.

In order to decide the next hotlink to be assigned, LOG maintains a vector \mathbf{w} , where $w(v), \forall v \in V$, gives the number of hotleaves that are descendants of v in the current tree. First, LOG assigns a hotlinks to a node u which splits the tree T into "balanced" subtrees. By balanced we mean that none of them has more than $\max\{t, (1 - t)\}|L|$ hotleaves, where t is a parameter whose value will be set in the analysis. Then, it updates the vector \mathbf{w} with respect to the improved tree and recursively handles each of these balanced trees. A pseudo-code for LOG

is presented in Figure 2. Figure 3 shows the trees T and $T^{(r,u)}$. In $T^{(r,u)}$, the subtrees $T_1 - T_u, T_2, \dots, T_k$ have at most $(1-t)|L|$ hotleaves while the subtrees rooted at the nodes that are children of u have at most $t|L|$ hotleaves.

```

LOG(T: directed tree)
If  $T$  has exactly one hotleaf
    Add a hotlink from  $r$  to this leaf
Else If  $T$  has more than one hotleaf
    Find a node  $u$  in  $V$  such that  $w(u) \geq tw(r)$  and  $w(v) < tw(r)$  for every  $v \in S_u$ . (*)
    Add the hotlink  $(r, u)$  to  $T$  to obtain an improved tree  $T^{\{(r,u)\}}$ .
    For every ancestor  $v$  of  $u$  in  $T$ , update the value of  $w(v)$  with respect to  $T^{\{(r,u)\}}$ .
    Let  $K = \{v | v \text{ is a child of } r \text{ in } T^{\{(r,u)\}}\}$ 
    Let  $M = \{v | v \text{ is a child of } u \text{ in } T^{\{(r,u)\}}\}$ .
    For every  $v \in (K \cup M) - \{u\}$  do
        Execute LOG for the subtree of  $T^{\{(r,u)\}}$  which contains  $v$  and all of its
        descendants
    
```

Fig. 2. The LOG Algorithm

The next lemma, whose proof we defer for the extended version of the paper, guarantees that it is always possible to find a node u as in the line (*) of the pseudo-code.

Lemma 1. Let T be a directed tree with root r and subtrees T_1, \dots, T_k rooted at children of r . Let $t \leq 1$ and $1/|L| \leq t \leq 1$. Then there exists a node $u \in T$ such that $w(u) \geq tw(r)$ and $w(v) < t.w(r)$ for every $v \in S_u$.

For this algorithm, we have the following theorem.

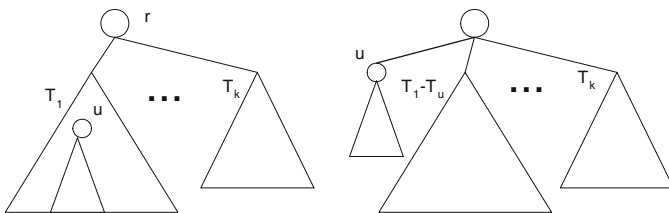


Fig. 3. The improved tree obtained by LOG due to the addition of (r, u)

Theorem 1. Let A be a directed tree with root r and subtrees T_1, \dots, T_k rooted at children of r . Let $t \leq 1$ and $1/|L| \leq t \leq 1$. Then there exists a node $u \in T$ such that $w(u) \geq tw(r)$ and $w(v) < t.w(r)$ for every $v \in S_u$. Let T^A be the tree obtained by adding the hotlink (r, u) to T . Then $h(T^A) \leq c \log w(r) + 1$.

$$c = \max \left\{ \frac{2}{\log \left(\frac{1}{t} \right)}, \frac{1}{\log \left(\frac{1}{1-t} \right)} \right\}.$$

We prove it by induction on the value of $w(r)$. If $w(r) = 1$ then $h(T^A) = 1 = c \log w(r) + 1$. Now, let us assume that the result holds for the case where $w(r) < k$. Under this assumption, we shall prove that it also holds for the case where $w(r) = k$. By the choice of u , we have in $T^{\{(r,u)\}}$: $w(u) \geq tw(r)$, $w(v) < tw(r)$ for every $v \in M$, and $w(v) \leq (1-t)w(r)$ for every child $v \in K - \{u\}$. By applying the inductive hypothesis on the subtrees rooted at the nodes in $(K \cup M) - \{u\}$, we obtain that

$$\begin{aligned} h(T^A) &= \max\{\max\{1 + h(T_v^A) \mid v \in K - \{u\}\}, \max\{2 + h(T_v^A) \mid v \in M\}\} \\ &\leq \max\{1 + c \log((1-t)w(r)) + 1, 2 + c \log(tw(r)) + 1\} \\ &= c \log w(r) + 1 + \max\left\{1 - c \log\left(\frac{1}{1-t}\right), 2 - c \log\left(\frac{1}{t}\right)\right\}. \end{aligned} \tag{1}$$

We establish this theorem by replacing the second and the third occurrence of c in the righthand side of the previous equality respectively by $1/\log(1/(1-t))$ and $2/\log(1/t)$. ■

From elementary calculus, we obtain that the best upper bound given by the previous theorem is achieved with $t = \frac{3-\sqrt{5}}{2}$. In this case, we have $c = \frac{2}{1-\log(3-\sqrt{5})} < 1.441$, which leads to the following corollary.

Corollary 1. $h^*(T) \leq h(T^A) \leq 1.441 \log m + 1$

Now, let us assume that the LOG algorithm uses the vector \mathbf{w} to maintain the number of nodes in the subtree rooted at each node rather than the number of hotleaves. We refer to this algorithm as the Modified LOG algorithm. In this case, we claim that it constructs a hotlink assignment A such that the height of T^A is not larger than $1.441 \log n + 1$. The proof for this claim is analogous to that of Theorem 1. The next theorem uses this claim to give an upper bound on the minimum height of a tree improved by a WC-optimal hotlink assignment.

Theorem 2. *Let T be a tree with m hotleaves and n nodes. Let A' be a WC-optimal hotlink assignment for T . Then, there exists a WC-optimal hotlink assignment A^* such that $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$.*

Let A' be an WC-optimal hotlink assignment for T . We prove this theorem constructing another WC-optimal hotlink assignment A^* from A' such that $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$. First, we insert in A^* every hotlink $(u, v) \in A'$ such that u is at a level smaller than $h^*(T)$ in $T^{A'}$. At this point, we already have $h(T^{A^*}) = h^*(T)$. Hence, by corollary 1, we obtain that any non-trivial subtree of T^{A^*} rooted at a level not smaller than $\lfloor 1.441 \log m + 1 \rfloor$ has no hotleaf and no hotlink assigned from. Since any change in such subtrees does not affect the optimality of A^* , we apply the Modified LOG algorithm on every subtree $T_v^{A^*}$ of T^{A^*} such that v is at level $\lfloor 1.441 \log m + 1 \rfloor$. Recall that the Modified LOG algorithm always construct improved trees with heights not greater than $1.441 \log n + 1$. Thus, by inserting the obtained hotlinks in A^* , we obtain that $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$, and the proof is done. ■

3.2 Lower Bounds

Here, we prove a simple lower bound on $h^*(T)$. First, observe that the maximum node outdegree in T^{A^*} is at most $d + 1$, where d is the maximum node outdegree of T . Then, we have the following lower bound.

Lemma 2. *Let T be a tree with m nodes and maximum node outdegree d . Then, $h^*(T) \geq \log_{d+1} m$.*

This lower bound may be very loose if d is close to m . In order to analyze the approximation algorithm proposed in the next section, we need a stronger lower bound. For that, we combine the previous lower bound with the fact that we can use subtrees of T to provide lower bounds on $h^*(T)$.

Theorem 3. *Let $T' = (V', E')$ be a subtree of T with root r . Then, $h^*(T) \geq \log_3 |V' \cap L|$.*

Let $T_2 = (V_2, E_2)$ be the tree induced by the ancestors of $V' \cap L$ in T . Applying Lemma 2 to T_2 , we get that $h^*(T_2) \geq \log_3 |V' \cap L|$. Thus, it remains to show that $h^*(T) \geq h^*(T_2)$. Let A^* be a WC-optimal hotlink assignment for T and let $A = \{(u, v) \in A^* \mid u, v \in V_2\}$. For every node $i \in V' \cap L$, we have that the level of i in T_2^A and T^{A^*} are the same. Hence, we obtain that $h^*(T) \geq h(T_2^A) \geq h^*(T_2)$, which completes our proof. ■

4 An Approximation Algorithm

In this section, we describe a $(14/3)$ -approximation algorithm for the WCHS problem that executes in $O(n \log m)$. Let us refer to this algorithm as the ALOG (Approximation LOG) algorithm. ALOG is similar to LOG except for the information stored by vector \mathbf{w} . Here, $w(v)$ stores the number of hotleaves of the binary subtree rooted at v which contains the largest number of hotleaves. Note that if v has only one child a , then $w(v) = w(a)$. Otherwise, $w(v) = w(a) + w(b)$ where a and b are the children of v with maximum \mathbf{w} 's value, that is, $w(v) = \max_{\{(a,b) \in S_v \times S_v \mid a \neq b\}} \{w(a) + w(b)\}$.

Now, we analyze the approximation ratio of ALOG. For that, we use $w(v)$ to denote the value of $w(v)$ with respect to T . As a consequence of Theorem 3 and the definition of \mathbf{w} , we have that

$$h^*(T) \geq \log_3 w(r). \tag{2}$$

Now, we develop an upper bound on $h(T^A)$. First, we must understand how \mathbf{w} evolves when the tree T is improved to become the tree $T^{\{(r,u)\}}$. The next lemma address this issue. For every $v \in V$, let $w^{T^{\{(r,u)\}}}(v)$ be the value of $w(v)$ with respect to $T^{\{(r,u)\}}$.

Lemma 3. Let K and M be disjoint sets of nodes, $u \in K$, $s_1 \in M$, and T be a hotlink assignment with $t \geq 0.5$.

- (.) $w^{T\{(r,u)\}}(v) \leq (1-t)w(r), \forall v \in K - \{u, s_1\}$
- (.) $w^{T\{(r,u)\}}(v) < tw(r), \forall v \in M$
- (.) $w^{T\{(r,u)\}}(s_1) \leq 2(1-t)w(r)$

We defer it for the extended version of this paper. ■

The following lemma gives an upper bound on the $h(T^A)$.

Lemma 4. Let A be a hotlink assignment, r be a node, and T be a hotlink assignment. Then $h(T^A) \leq c \log w(r) + 1$, where $c = \max \left\{ \frac{2}{\log(\frac{1}{t})}, \frac{1}{\log(\frac{1}{2(1-t)})} \right\}$.

The proof is analogous to that of Theorem 1. The only difference is that we use the upper bound given by Lemma 3 for the child of r that is ancestor of u , when the hotlink (r, u) is assigned. ■

Again, we obtain from elementary calculus that the optimal value of t is $\frac{9-\sqrt{17}}{8}$ (observe that $t > 0.5$ as required in Lemma 3). In this case, we have $c = \frac{2}{3-\log(9-\sqrt{17})} < 2.801$, which leads to the following theorem.

Theorem 4. $h(T^A) \leq \lceil \log_3 w(r) \rceil + 14/3$.

If $w(r) = 1$ then ALOG always constructs an optimal hotlink assignment. Hence, let us assume that $w(r) > 1$. It follows from equation (2) and Lemma 4 that

$$\frac{h(T^A)}{h^*(T)} \leq \frac{c \log w(r) + 1}{\log_3 w(r)}. \tag{3}$$

However, since the objective function for the WCHS problem is always integer, we also have that

$$\frac{h(T^A)}{h^*(T)} \leq \frac{\lfloor c \log w(r) + 1 \rfloor}{\lceil \log_3 w(r) \rceil}. \tag{4}$$

We establish this theorem by using (3) whenever $w(r) \leq 125$ and (4) otherwise. In the first case, we verify through a computer that the theorem holds for all possible values of $w(r)$. In the second case, we have that

$$\frac{h(T^A)}{h^*(T)} \leq \frac{c \log w(r)}{\log_3 w(r)} + \frac{1}{\log_3 126} < 14/3.$$

■

Now, we analyze the running time of the ALOG algorithm.

Theorem 5. *For a tree T with n nodes and m hotlinks, the algorithm runs in $O(n \log m)$ time.*

(Sketch) In order to calculate the value of vector w for a tree T , the algorithm takes linear time. Then, it is recursively executed to solve subproblems defined by trees for which the w values of their roots is at a constant factor of the w -value for T 's root. This fact, is assured by Lemma 3 Thus, one can write a recursion relation which proves that ALOG runs in $O(m \log n)$ ■

5 An Exact Algorithm

In this section, we introduce the PATH algorithm, an exact dynamic programming based algorithm for solving hotlinks assignment problems. PATH runs in $O(n(nm)^{2.284})$ time and uses $O(n(nm)^{1.441})$ space.

In order to execute PATH we must provide an integer D . For a given D , PATH spends $O(n \cdot 3^D)$ to find a hotlink assignment A_D^* (if it exists) with the following properties: $H(T^{A_D^*}) \leq D$ and for every feasible hotlink assignment A , with $H(T^A) \leq D$, we have $h(T^{A_D^*}) \leq h(T^A)$.

In other words, PATH finds the best possible assignment under the constraint that the tree produced by such assignment has height at most D . Since Theorem 2 assures that there is a WC-optimal hotlink assignment A^* such that $H(T^{A^*}) \leq 1.441(\log m + \log n) + 2$, we execute PATH with $D = 1.441(\log m + \log n) + 2$. This assures that PATH produces in polynomial time an optimal solution for the WCHS problem.

5.1 The P-WCHS Problem

Observe that the WCHS problem could be exactly solved as follows. For each possible hotlink assignment (r, u) from r , obtain the improved tree $T^{\{(r,u)\}}$. Then, for each child v of r in $T^{\{(r,u)\}}$, recursively solve the subproblem where the input tree is the subtree rooted at v . At the end, return the best solution found. This approach, however, generates an exponential number of subproblems.

In order to explain the strategy used by PATH, let T be a tree rooted at r and let f be the rightmost child of r . Recall that T_f is the largest subtree of T rooted at f . First, PATH partitions the set of feasible hotlink assignments into two sets: the set of assignments in which a hotlink is assigned from r to some node of T_f and the set of assignments in which it does not occur. Let A_1 and A_2 be the best possible assignments that belong to the first and the second set, respectively. Clearly, $h^*(T) = \min\{h(T^{A_1}), h(T^{A_2})\}$. For a given tree \mathcal{T} , let $g^*(\mathcal{T})$ be the optimal solution for the WCHS problem, with input \mathcal{T} , constrained to assignments where no hotlink leaves the root of \mathcal{T} . Being $T_f \cup \{r\}$ the tree induced by both r and the descendants of f in T , it is not difficult to check that $h(T^{A_1}) = \max\{h^*(T - T_f), g^*(T_f \cup \{r\})\}$ and $h(T^{A_2}) = \max\{h^*(T_f \cup \{r\}), g^*(T - T_f)\}$. Thus, we have

$$h^*(T) = \min\{\max\{h^*(T - T_f), g^*(T_f \cup \{r\})\}, \max\{h^*(T_f \cup \{r\}), g^*(T - T_f)\}\}.$$

The discussion above is for showing that in order to solve the WCHS problem through this partitioning approach we must be able to solve a generalized problem

in which some nodes may be forbidden, that is, no hotlink can leave them. Such a problem, denoted by P-WCHS, is formalized below:

Input:

- i) a directed path $\mathbf{q} = (V_{\mathbf{q}}, E_{\mathbf{q}})$ where $V_{\mathbf{q}} = \{q_1, \dots, q_k\}$ and $E_{\mathbf{q}} = \{(q_i, q_{i+1}) \mid 1 \leq i \leq k-1\}$;
- ii) a binary vector $\mathbf{a} = (a_1, \dots, a_k, a_{k+1}, b) \in \{0, 1\}^{k+2}$;
- iii) a tree $T^s = (V^s, E^s)$ rooted at r .
- iv) an integer D

Output: A hotlink assignment A to the tree $T_{\mathbf{q}} = (V_{\mathbf{q}} \cup V^s, E_{\mathbf{q}} \cup E^s \cup \{(q_k, r)\})$, satisfying the following six conditions:

- (a) A is feasible in the sense of the WCHS problem;
- (b) No hotlink can point to a node in $V_{\mathbf{q}}$;
- (c) For $i = 1, \dots, k$, if $a_i = 0$, then no hotlink can leave q_i ;
- (d) If $a_{k+1} = 0$, then no hotlink can leave r ;
- (e) if $b = 0$, then no hotlink can point to r ;
- (f) $H(T_{\mathbf{q}}^A) \leq D$

Objective: Minimize the maximum level of a hotleaf in the resulting improved tree $T_{\mathbf{q}}^A$.

Observe that the WCHS problem is a particular case of the P-WCHS problem when \mathbf{q} is empty, $b = a_1 = 1$, and $D = H$. Thus, an exact algorithm for P-WCHS is also an exact algorithm for WCHS.

5.2 Solving P-WCHS

Figure 4 is used throughout this section to illustrate some ideas. Figure 4.(a) presents an instance of P-WCHS where the path \mathbf{q} consists of four nodes q_1, q_2, q_3 and q_4 . If a node $q_i \in \mathbf{q}$ is such that $a_i = 1$, then q_i is said to be *available*. The only non-available node, q_3 , is black colored. The subtree T^s is rooted at a node r which has three children. Since $a_5 = b = 1$, hotlinks can be assigned from and to the node r .

We need to introduce the following conventions: given a binary vector \mathbf{c} and a binary string d , we use $\mathbf{c}d$ to indicate the vector obtained by concatenating \mathbf{c} and d . For example, if $\mathbf{c} = (0, 1)$ and $d = 100$, then $\mathbf{c}d = (0, 1, 1, 0, 0)$ and $d\mathbf{c} = (1, 0, 0, 0, 1)$. We use \mathbf{c}_i to denote the vector obtained by truncating the i th first components of \mathbf{c} . At the previous example, $\mathbf{c}_1 = (0)$ and $\mathbf{c}_2 = (0, 1)$. For a directed path \mathbf{q} and a node u , we use $\mathbf{q} \rightarrow u$ to indicate the path obtained by inserting u at the end of \mathbf{q} . We use \mathbf{q}_i to denote the subpath of \mathbf{q} formed by its i -th first nodes.

Let $h^*(\mathbf{q}, \mathbf{a}, T^s)$ denote the cost of an optimal solution of a P-WCHS instance defined by a binary vector \mathbf{a} , a directed tree T^s , a path \mathbf{q} with $|\mathbf{a}| - 2$ nodes and an integer D . If $|\mathbf{q}| > D$, PATH set $h^*(\mathbf{q}, \mathbf{a}, T^s) = \infty$. Hence, let us assume that $|\mathbf{q}| \leq D$. In order to solve this instance we must consider the following cases:

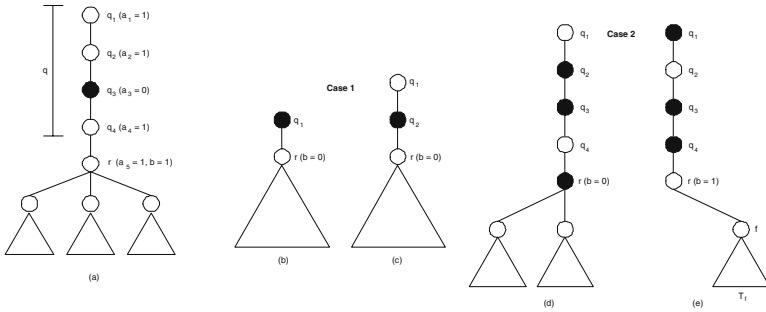


Fig. 4. (a) an instance of the P-WCHS problem. (b) and (c) two possible subproblems generated in the case 1. (d) and (e) the decomposition in the case 2 when $c = (0, 1, 0, 0, 1)$

Case 1: some hotlink is assigned from a node of \mathbf{q} to r in the optimal solution;
 Case 2: no hotlink is assigned from a node of \mathbf{q} to r in the optimal solution.

Case 1. This case is only considered when $b = 1$. In this case, we must assign a hotlink from some available node to r . Thus, we have $\sum_{i=1}^k a_i$ possibilities. If (q_1, r) is assigned to the tree of Figure 4.(a), then PATH generates the subproblem of Figure 4.(b). In fact, the addition of hotlink (q_1, r) creates an improved tree where q_1 has two children: T^s and the path (q_2, q_3, q_4) . However, since q_2, q_3 and q_4 are not ancestors of hotleaves, they can be removed without modifying the cost of the solution. Observe that b is set to 0, otherwise we would generate a solution with crossing arcs. When (q_2, r) is assigned, instead of (q_1, r) , then PATH generates the subproblem of Figure 4.(c). In general, if some hotlink points to r in the optimal solution, we have that

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \min_{i \in \{1, 2, 3, \dots, k\} \mid a_i = 1} \{h^*(\mathbf{q}_i, \mathbf{a}_{i-1} 0 a_{k+1} 0), T^s\} \tag{5}$$

Case 2. In this case all the available nodes of \mathbf{q} may only point to some node in $V_r - \{r\}$. Thus, PATH must decide which of the available nodes are allowed to point to the nodes of T_f , the subtree rooted at the last child f of r (assuming any order). Let $k' = \sum_{i=1}^{k+1} a_i$ be the number of available nodes. Then, PATH has $2^{k'}$ possibilities to take such a decision. Since it is not clear which one is the best, then all of them are considered. In order to clarify this idea, let us consider the possibility where q_2 and r remain available for T_f (see Figure 4.(e)). As a consequence, only q_1 and q_4 will be allowed to point to nodes in $T^s - T_f$ (Figure 4.(d)). Figure 4.(d) defines a new subproblem $(\mathbf{q}, \mathbf{a}', T^s - T_f)$, where $\mathbf{a}' = (1, 0, 0, 1, 0, 0)$. Note that b is set to 0 since we are in case 2. On the other hand, Figure 4.(e) defines a new subproblem $(\mathbf{q} \rightarrow r, \mathbf{a}'', T_f)$, where $\mathbf{a}'' = (0, 1, 0, 0, 1, 1)$.

Thus, the maximum between the cost of the optimal solutions for the subproblems defined by Figures 4.(d) and 4.(e) is the cost of the optimal solution for the problem of Figure 4.(a) under the assumptions that no hotlink can be

assigned to r (Case 2), the nodes q_2 and r cannot point to nodes in $T^s - T_f$, and the nodes q_1 and q_4 cannot point to nodes in T_f .

In general, let C be a set of binary vectors defined by $C = \{(c_1, \dots, c_{k+1}) \mid c_i \leq a_i \text{ for } i = 1, \dots, k+1\}$. Each $\mathbf{c} \in C$ corresponds to one of the $2^{k'}$ possibilities for selecting the nodes that will remain available to point to nodes in T_f . Furthermore, let $\bar{\mathbf{c}} = \mathbf{a} - \mathbf{c}$. This vector defines which nodes from \mathbf{q} will remain available to point to nodes in $T^s - T_f$. Then, by considering all choices for \mathbf{c} , we have that

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \min_{\mathbf{c} \in C} \{ \max\{h^*(\mathbf{q} \rightarrow r, \mathbf{c}11, T_f), h^*(\mathbf{q}, \bar{\mathbf{c}}0, T^s - T_f)\} \}, \quad (6)$$

under the assumption that no node from \mathbf{q} points to r .

Cases 1 and 2 together: Let P and Q be respectively, the righthand side of equations (5) and (6). Thus,

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \begin{cases} Q & \text{if } b = 0 \\ \min\{P, Q\} & \text{if } b = 1 \end{cases}$$

Stop Conditions: If T^s has only one node and this node is a hotleaf, say l , then the best thing to do is to assign a hotlink from the first available node in \mathbf{q} to l . Thus,

$$h^*(\mathbf{q}, \mathbf{a}, T^s) = \begin{cases} \min\{i \mid 1 \leq i \leq k \text{ and } a_i = 1\} & \text{if } \mathbf{q} \text{ has some available node} \\ k, & \text{otherwise} \end{cases} \quad (7)$$

If T^s has no hotleaves then $h^*(\mathbf{q}, \mathbf{a}, T^s) = 0$.

5.3 Analysis

In order to prove that the algorithm is correct, we must argue that if A is a feasible hotlink assignment with $H(T^A) \leq D$, then A is among the assignments considered by the PATH algorithm. Since PATH only discards subproblems for which $|\mathbf{q}| > D$, it is enough to show that none of the subproblems generated by PATH to produce the hotlink assignment A has this property. Let $\mathbf{I}' = (\mathbf{q}', \mathbf{a}', T')$ be one of these subproblems. By observing that \mathbf{q}' is also a path in T^A , we conclude that $|\mathbf{q}'| \leq D$.

Now, we analyze both the time and space complexities of the PATH algorithm.

Theorem 6. *Let T be a tree with n nodes and D hotleaves. The PATH algorithm generates $O(n3^D)$ subproblems and uses $O(n2^D)$ space.*

First, let us analyze the number of generated subproblems. By means of the dynamic programming technique, this number has the same order as the PATH space usage. For each node $u \in V$ and each integer $q \in \{0, 1, \dots, |S_u| - 1\}$, PATH generates a subtree T^s by removing from T_u all the subtrees rooted at the last q children of u . Now, let r be the root of T . Observe that PATH generates

exactly m trivial subtrees. Moreover, each node $v \in V - \{r\}$ is the last (non-removed) child of the root in exactly one generated non-trivial subtree. Hence, PATH generates $m + n - 1 = O(n)$ subtrees. For each generated subtree, PATH generates all possible paths vectors \mathbf{a} . Since we have exactly 2^{i+2} possible vectors \mathbf{a} corresponding to a path \mathbf{q} with i nodes, the number of generated vectors is given by $\sum_{i=0}^D 2^{i+2} = O(2^D)$.

As a result, we have that PATH uses $O(n2^D)$ space. Finally, we obtain the time complexity of PATH by counting the number of subproblems checked to calculate each value of $h^*(\mathbf{q}, \mathbf{a}, T^s)$. Let r^s be the root of T^s . In the Case 1, PATH checks $O(D)$ subproblems, since we have $O(D)$ possible hotlinks from a node in \mathbf{q} to r^s . On the other hand, in Case 2, the number of subproblems checked by PATH depends on the number of available hotlinks in both \mathbf{q} and r^s . For j available hotlinks, $O(2^j)$ subproblems are checked, since this is the complexity of the number of possible distributions for these hotlinks between two subproblems. Moreover, we have $n \binom{D+2}{j}$ subproblems with j available hotlinks, for $j = 0, 1, \dots, D + 2$. Hence, we have that the time complexity of PATH is given by

$$O(n2^D D) + \sum_{j=0}^{D+2} O\left(n \binom{D+2}{j} 2^j\right) = O(n2^D D) + O(n(2+1)^{D+2}) = O(n3^D). \blacksquare$$

Corollary 2. *Let n and m be the number of nodes and hotlinks, respectively, in a graph. Then, the time complexity of the PATH algorithm is $O(n(nm)^{2.284})$ and the space complexity is $O(n(nm)^{1.441})$.*

Theorem 2 assures that there is an optimal solution for WCHS with height at most $1.441(\log m + \log n) + 2$. This assures that PATH can be executed with $D = 1.441(\log m + \log n) + 2$, which implies on the complexities above. \blacksquare

References

1. Gerstel, Kutten, Matchin, Peleg: Hotlink enhancement algorithms for web directories (extended abstract). In: ISAAC: 14th International Symposium on Algorithms and Computation. (2003)
2. Perkowitz, M., Etzioni, O.: Towards adaptive Web sites: conceptual framework and case study. *Computer Networks* (Amsterdam, Netherlands: 1999) **31** (1999) 1245–1258
3. Bose, P., Kranakis, E., Krizanc, D., Martin, M.V., Czyzowicz, J., Pelc, A., Gasieniec, L.: Strategies for hotlink assignments. In: *International Symposium on Algorithms and Computation*. (2000) 23–34
4. Fuhrmann, S., Krumke, S.O., Wirth, H.C.: Multiple hotlink assignment. In: *Proceedings of the Twenty-Seventh International Workshop on Graph- Theoretic Concepts in Computer Science*. (2001)

5. Kranakis, E., Krizanc, D., Shende, S.: Approximate hotlink assignment. In: ISAAC: 12th International Symposium on Algorithms and Computation. (2001)
6. Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Enhancing hyperlink structure for improving web performance. *Journal of Web Engineering* **1** (2003) 93–127
7. Matichin, Peleg: Approximation algorithm for hotlink assignments in web directories. In: WADS: 8th Workshop on Algorithms and Data Structures. (2003)
8. Pessoa, A., Laber, E., Souza, C.: Efficient implementation of hotlink assignment algorithms for web sites. In: Proceedings of ALENEX. (2004)
9. Matichin, Peleg: Approximation algorithm for hotlink assignment in the greedy model. In: SIROCCO 2004:11th Colloquium on Structural Information and Communication Comple. (2004)

Dynamic Tree Cross Products



Marcus Raitner

University of Passau, D-94032 Passau, Germany

Marcus.Raitner@Uni-Passau.De

Abstract. *Range searching over tree cross products* – a variant of classic range searching – recently has been introduced by Buchsbaum et al. (*Proc. 8th ESA*, vol. 1879 of *LNCS*, pp. 120–131, 2000). A tree cross product consist of hyperedges connecting the nodes of trees T_1, \dots, T_d . In this context, range searching means to determine all hyperedges connecting a given set of tree nodes. Buchsbaum et al. describe a data structure which supports, besides queries, adding and removing of edges; the tree nodes remain fixed. In this paper we present a new data structure, which additionally provides insertion and deletion of leaves of T_1, \dots, T_d ; it combines the former approach with a novel technique of using search trees superimposed over ordered list maintenance structures. The extra cost for this dynamization is roughly a factor of $\mathcal{O}(\log n / \log \log n)$. The trees being dynamic is especially important for *maintaining hierarchical graph views*, a problem that can be modeled as tree cross product. Such views evolve from a large base graph by the contraction of subgraphs defined recursively by an associated hierarchy. The graph view maintenance problem is to provide methods for refining and coarsening a view. In previous solutions only the edges of the underlying graph were dynamic; with the application of our data structure, the node set becomes dynamic as well.

1 Motivation

Many graphs, such as network traffic graphs, the web graph, or biochemical pathways [1], are too large to display or edit them effectively. A well-established technique to solve this problem is to partition the graph recursively into a hierarchy of subgraphs. The complete road map of Europe, for instance, is a rather large graph; a hierarchy on it can be defined, for instance, by grouping places and roads within the same city, then the cities within the same state, and so on. Not every city or state is always needed in full detail; the dispensable subgraphs, therefore, are contracted into a single , representing the city or state as a whole. Edges from within the contracted subgraph to nodes outside are retained as edges from the meta node to the outside place. This leads to an abstract representation of the graph, a , which is very convenient, because both an overview of the whole graph and the necessary details are displayed simultaneously.

In an interactive scenario, this facilitates exploring and editing a large graph: the user can choose which subgraphs to contract into meta nodes and which

to expand, i.e., to replace with its subordinate subgraphs. Depending on the admissible modifications of the hierarchy and the graph, Buchsbaum and Westbrook [2] differentiate three variants of this problem: in the *static* case the graph and the hierarchy are both fixed; in the *dynamic* variant graph edges can be inserted or deleted; in the *dynamic hierarchical* variant the graph additionally is subject to node insertions and deletions, and the hierarchy may change through splitting and merging of clusters.

As shown in [3] and briefly recapitulated in Sect. 3, maintaining hierarchical graph views can be formulated as a special case of *range searching*. A tree cross product consists of disjoint trees T_1, \dots, T_d and a set of d -dimensional hyperedges $\mathbf{u} = (u_1, \dots, u_d)$ with $u_i \in T_i$ for $i = 1, \dots, d$. In this context, range searching means to decide efficiently whether there is a hyperedge between the subtrees of given tree nodes and to report all those hyperedges. In [3] the set of hyperedges is dynamic, but the trees are static; we generalize this approach and allow insertion and deletion of leaves of T_1, \dots, T_n as well. After a formal definition of the problem, the data structure for the two-dimensional tree cross product is described in Sect. 2.1, which then serves as basis for the recursive description of the higher dimensional case in Sect. 2.2.

In Sect. 3 we introduce the new *dynamic hierarchical* variant of the graph view maintenance problem for *dynamic hierarchical graphs* [4], a more general model of hierarchically structured graphs than the *dynamic graphs* [5] used in previous approaches [2,3]. Formulated as a tree cross product, this problem can be solved efficiently with our data structure. The new dynamic leaves variant extends the dynamic graph variant by insertion and deletion of graph nodes, i.e., leaves of the hierarchy. In contrast to the dynamic graph and tree variant, it lacks splitting and merging of clusters. Thus, it is adequate for dynamic graphs with a fixed hierarchical structure, such as network traffic graphs: computers are identified by their IP addresses, edges represent traffic, and the hierarchy is given by the structure of the IP addresses. The hierarchy is fixed, because the structure of an IP address does not change. Further examples are road maps or biochemical reaction networks [1], which consist of all reactions in some organism; the hierarchy is often defined by grouping reactions that belong to the same biochemical pathway, e.g., the citric acid cycle. Unable to handle graphs with a dynamic node set, previous solutions [2,3] are not appropriate models for these applications.

2 Range Searching over Tree Cross Products

Let $T = (V(T), E(T))$ be a rooted tree with node set $V(T)$ and edge set $E(T)$. For a node $v \in V(T)$, let $\text{children}(v)$ denote the set of all children of v and $\text{parent}(v)$ the parent of v . The *descendants* of v , $\text{desc}(v)$, are all nodes in the subtree rooted at v . Conversely, v is an *ancestor* of each $u \in \text{desc}(v)$. Given d disjoint, rooted trees T_1, \dots, T_d , consider a d -partite hypergraph G such that $V(G) = \bigcup_{i=1}^d V(T_i)$ and $E(G) \subseteq \prod_{i=1}^d V(T_i) = V(T_1) \times \dots \times V(T_d)$. We define $n = |V(G)|$, $m = |E(G)|$, and $D = \max_{i=1}^d \text{depth}(T_i)$. For a tuple $\mathbf{u} = (u_1, \dots, u_d) \in \prod_{i=1}^d V(T_i)$, let

$$E(\mathbf{u}) = \{\mathbf{x} = (x_1, \dots, x_d) \in E(G) \mid \forall 1 \leq i \leq d : x_i \in \text{desc}(u_i)\}$$

be the set of hyperedges between descendants of \mathbf{u} 's elements; see Fig. 1 for an example of a two-dimensional tree cross product. We call a tuple \mathbf{u} an *induced tuple* if $E(\mathbf{u}) \neq \emptyset$; the *induced tuple set* \mathcal{I} consists of the node set $V(\mathcal{I}) = V(G)$ and all induced hyperedges: $E(\mathcal{I}) = \{\mathbf{u} \in \prod_{i=1}^d V(T_i) \mid E(\mathbf{u}) \neq \emptyset\}$. In the example of Fig. 1, for instance, the set $E(v_1, u_2)$ contains three edges, but $E(u_1, u_2)$ is empty; this yields an induced edge $(v_1, u_2) \in E(\mathcal{I})$, but none between u_1 and u_2 .

As defined in [3], the *induced tuple set* \mathcal{I} is to perform the following operations on tuples $\mathbf{u} \in \prod_{i=1}^d V(T_i)$:

- **edgeQuery**(\mathbf{u}) determines if $E(\mathbf{u}) \neq \emptyset$, i. e., whether \mathbf{u} is induced,
- **edgeReport**(\mathbf{u}) determines the set $E(\mathbf{u})$,
- **edgeExpand**(\mathbf{u}, j), where $1 \leq j \leq d$; determines all induced hyperedges $(u_1, \dots, u_{j-1}, x, u_{j+1}, \dots, u_d) \in E(\mathcal{I})$, where $x \in \text{children}(u_j)$.

Besides inserting and deleting hyperedges (**newEdge**(\mathbf{u}) and **deleteEdge**(\mathbf{u})), our data structure supports the following new operations for adding and removing leaves:

- **addLeaf**(u), where $u \in V(T_j)$; adds a new leaf to T_j as child of node u ,
- **deleteLeaf**(u), where u is a leaf in $V(T_j)$ and u is not the root of T_j ; removes u and all hyperedges incident to it from T_j .

2.1 The Two-Dimensional Case

With [3] we share the idea to order the nodes of each tree T_1 and T_2 linearly such that for each tree node v the set $\text{desc}(v)$ is fully determined by $\min(v)$ and $\max(v)$, the smallest and the largest node in the subtree rooted at v . Clearly, traversing each tree in post-order yields such an order.

Since **addLeaf**(u) was defined to insert the new leaf as a child of u , insertions occur at any point in the linear order. Therefore, simply assigning consecutive integers to the nodes, as in [3], is inefficient, because all nodes following the new one have to be renumbered. There are already efficient data structures for the problem of performing **order** queries on an ordered list that is subject to **insert** and **delete** operations [6,7]. All of them assign a numerical label (often integers) to the elements, making the **order** query a simple comparison of the labels. The most efficient solution allows all operations to be performed in $\mathcal{O}(1)$ worst-case time [7, Sect. 3]. Using a technique for insertions and deletions in dense sequential files [8] on the top level of a four-level data structure, it is, however, rather complicated; a simpler, slightly less efficient, data structure might be more suitable for an implementation. For example, [6] and [7, Sect. 2] both describe solutions with $\mathcal{O}(1)$ worst-case time for **insert** and **delete** and $\mathcal{O}(1)$ worst-case time for **order**. Since our approach will treat the order maintenance component as a black box, any will do, yet only with an $\mathcal{O}(1)$ worst-case solution the time bounds of Theorems 1 and 2 are worst-case. Besides, we need access to the successor and predecessor of a node in this order. If not already part of the

order maintenance structures, we separately maintain a doubly linked list of the nodes for each tree. In the following, we use $<$, \leq , \geq , and $>$ to compare two nodes instead of the corresponding `order` query.

We keep the list of children at a tree node sorted according to the respective order for the tree. In addition to the values $\min(u_1)$ and $\max(u_1)$, we store at each tree node $u_1 \in V(T_1)$ (and symmetrically for the nodes $u_2 \in V(T_2)$) the set

$$S(u_1) = \{u_2 \in V(T_2) \mid \exists(u'_1, u_2) \in E(G) : u'_1 \in \text{desc}(u_1)\},$$

i. e., all nodes of the tree T_2 that are connected to a node in the subtree of u_1 ; see Fig. 1. Although defined as a set, $S(\cdot)$ intrinsically is a multiset: it stores multiple entries of the same node, because the entries correspond to edges, and an `edgeReport` needs to find all of them. A node $u'_2 \in S(u_1) \cap \text{desc}(u_2)$ indicates an edge $(u'_1, u'_2) \in E(G)$ for some $u'_1 \in \text{desc}(u_1)$; if no such node exists, no edge connects $\text{desc}(u_1)$ and $\text{desc}(u_2)$. Therefore, `edgeQuery`(u_1, u_2) can be implemented by checking whether $S(u_1) \cap \text{desc}(u_2) = \emptyset$.

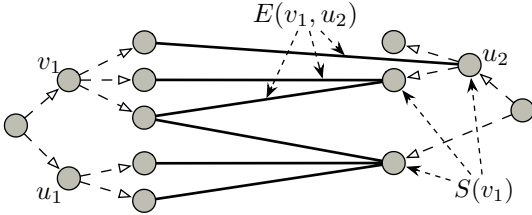


Fig. 1. Example of a two-dimensional tree cross product. The dashed edges belong to the two trees; the solid ones are the hyperedges $E(G)$

For the sets $S(\cdot)$ we need a data structure that, besides `insert` and `delete`, efficiently supports the `succ` operation `succ`: for $u_1 \in V(T_1)$ and $u_2 \in V(T_2)$, `succ`($S(u_1), u_2$) returns the smallest $v \in S(u_1)$ with $v \geq u_2$ or `null` if no such element exists. Observe that $S(u_1) \cap \text{desc}(u_2) \neq \emptyset$ if and only if `succ`($S(u_1), \min(u_2)$) $\leq \max(u_2)$; thus, `edgeQuery`(u_1, u_2) returns true if and only if `succ`($S(u_1), \min(u_2)$) $\leq \max(u_2)$ [3]. We maintain each set $S(\cdot)$ as a balanced search tree with respect to the order provided by the corresponding order maintenance data structure. Hence, `insert`, `delete`, and `succ` can be done in $\mathcal{O}(\log n)$ worst-case time, provided that the `order` operation is $\mathcal{O}(1)$ worst-case time. Additionally, the leaves of the search trees are linked to facilitate the `edgeReport` operation.

Instead of balanced search trees, in [3] contracted stratified trees (CST) [9] are used for the sets $S(\cdot)$. These improve the time bounds for `insert`, `delete`, and `succ` from $\mathcal{O}(\log n)$ to $\mathcal{O}(\log \log n)$ and increase the required space by a factor of $\mathcal{O}(\log \log n)$; both effects can be seen in Table 1. A CST, however, stores a subset of a integer universe; this is impractical here, because the sets $S(\cdot)$ are subsets of the set of tree nodes, which is – in contrast to [3] –

Table 1. Summary of results and comparison for the d -dimensional data structure; all bounds are worst-case. Let $D = \max_{i=1}^d \text{depth}(T_i)$. For `edgeReport` and `edgeExpand`, k denotes the size of the output. The `edgeQuery` and `edgeReport` bounds stated in [3] do not include the additive d terms, but they seem unavoidable given the description

	Approach of [3]	Our data structure
Space	$\mathcal{O}(m(2D)^{d-1} \log \log n)$	$\mathcal{O}(m(2D)^{d-1})$
<code>edgeQuery(u)</code>	$\mathcal{O}(d + \log \log n)$	$\mathcal{O}(d + \log n)$
<code>edgeReport(u)</code>	$\mathcal{O}(\log \log n + k)$	$\mathcal{O}(\log n + k)$
<code>edgeExpand(u, j)</code>	$\mathcal{O}(d + k \log \log n)$	$\mathcal{O}(d + k \log n)$
<code>newEdge(u)</code>	$\mathcal{O}((2D)^{d-1} \log \log n)$	$\mathcal{O}((2D)^{d-1} \log n)$
<code>deleteEdge(u)</code>	$\mathcal{O}((2D)^{d-1} \log \log n)$	$\mathcal{O}((2D)^{d-1} \log n)$
<code>addLeaf(u)</code>	n/a	$\mathcal{O}(D)$
<code>deleteLeaf(u)</code>	n/a	$\mathcal{O}(D)$

dynamic. Apart from the universe not being fixed, using the integer labels of the order maintenance structures [6, 7] in a CST is complicated: during an `insert` operation, nodes following the inserted one are possibly shifted, i. e., get a new number. Hence, all sets $S(\cdot)$ containing a shifted node need to be updated.

In order to efficiently perform the `edgeExpand` operation (see Algorithm 1), we need to determine the ancestor of a node at a given depth of the tree. This is well studied both in the static [10, 11] and the dynamic variant [12, 13]. Since we need to add and remove leaves, we will use the dynamic data structure described in [12]. It preprocesses a tree in linear time and space such that level ancestor queries and adding leaves can be performed in $\mathcal{O}(1)$ worst-case time [12, Theorem 6]. Deleting leaves is not explicitly mentioned in [12], but it is obviously possible in constant time by simply deleting the leaf. The space bound, however, would no longer be linear in the number of tree nodes. Similar to maintaining dynamic arrays, we can rebuild the data structure when, for instance, half of the nodes have been deleted. This takes $\mathcal{O}(1)$ amortized cost per delete, but using the standard doubling technique we can distribute it over a sequence of operations such that every operation is worst-case $\mathcal{O}(1)$.

Altogether, our approach combines the idea of [3] with our novel technique of using search trees superimposed over order maintenance structures. This makes the previous data structure more dynamic in regard to insertion and deletion of leaves, while the slow-down for the other operations – roughly a factor of $\mathcal{O}(\log n / \log \log n)$ – is tolerable; see Table 1.

Lemma 1. *Let G be a graph with m edges and n nodes. The space needed to store all sets $S(w)$ for $w \in V(G)$ is $\mathcal{O}(mD \log n)$.*

Each edge $e = (u_1, u_2) \in E(G)$ can contribute an entry only to those sets $S(w)$ where w is an ancestor of either u_1 or u_2 . Therefore, the space needed for all sets $S(\cdot)$ together is $\mathcal{O}(mD)$. They are built as follows: for each edge

$e = (u'_1, u'_2) \in E(G)$, u'_1 is inserted into all sets $S(u_2)$, where u_2 is an ancestor of u'_2 (and symmetrically u'_2 into $S(u_1)$ for each ancestor u_1 of u'_1). These are $\mathcal{O}(mD)$ insert operations in balanced search trees, each of which takes $\mathcal{O}(\log n)$.

The additional space for any of the order maintenance data structures [6, 7] is linear in the number of elements they contain, i. e., $\mathcal{O}(n)$; preprocessing takes $\mathcal{O}(n)$ worst-case time. The level ancestor structure also can be preprocessed in linear time and space [12, Theorem 6]. □

Lemma 2. $\text{edgeQuery} \dots \text{edgeReport} \dots \mathcal{O}(\log n) \dots \mathcal{O}(\log n + k) \dots k \dots$

$\text{edgeQuery}(u_1, u_2)$ is done by checking whether $\text{succ}(S(u_1), \min(u_2)) \leq \max(u_2)$. Since the set $S(\cdot)$ is stored as a balanced search tree, the succ operation and thus the whole edgeQuery take $\mathcal{O}(\log n)$ worst-case time. Finding the first edge reported by the edgeReport is essentially an edgeQuery . Since the leaves of the search trees for the sets $S(\cdot)$ are linked, the remaining edges in $E(u_1, u_2)$ are discovered in constant time each. □

We can implement edgeExpand by an appropriate collection of edgeQuery operations; in general, however, this is less efficient than Algorithm 1, which is similar to [3]. Since it simplifies the description, Algorithm 1 treats the expansion of the first element of an edge only, i. e., $\text{edgeExpand}((u_1, u_2), 1)$; expanding the second element works analogously.

Algorithm 1: $\text{edgeExpand}((u_1, u_2), 1)$

input : $(u_1, u_2) \in E(\mathcal{I})$, i. e., an induced edge (u_1, u_2)
output: all children u'_1 of u_1 such that $(u'_1, u_2) \in E(\mathcal{I})$
let v_1, \dots, v_k be the ordered list of children of u_1
 $R \leftarrow \emptyset, t \leftarrow v_1$
repeat
 $s \leftarrow \text{succ}(S(u_2), \min(t))$
 if $s \leq \max(v_k)$ **then**
 if $s > \max(t)$ **then** set t to the ancestor of s on the level of children(u_1)
 $R \leftarrow R \cup \{t\}$
 if $t \neq v_k$ **then** advance t to the next child
 end
until $t = v_k$ or $s > \max(v_k)$
return R

Lemma 3. $\text{edgeExpand}((u_1, u_2), j) \dots \mathcal{O}(k \log n) \dots k \dots$

Without loss of generality, we assume $j = 1$; the case $j = 2$ is symmetric. Let v_1, v_2, \dots, v_k denote the children of u_1 , in ascending order according to the linear order of the tree T_1 , i. e., $v_1 < v_2 < \dots < v_k$. Note that the children are stored in this order and do not need to be sorted. We start with v_1 and determine whether it is connected to u_2 by calculating $s = \text{succ}(S(u_2), \min(v_1))$. If $s \leq \max(v_1)$, v_1 is reported; if $\max(v_1) < s \leq \max(v_k)$, the ancestor s'

of s among the children of u_1 is reported by way of the level ancestor data structure. This procedure is iterated until $s > \max(v_k)$; see Algorithm 1. Each `succ` operation, except the last, yields a new result. Since determining the level ancestor takes constant time, we get $\mathcal{O}(k \log n)$ worst-case time. \square

After adding a new edge (u'_1, u'_2) to $E(G)$, we insert u'_1 into $S(u_2)$ for all ancestors u_2 of u'_2 and u'_2 into $S(u_1)$ for the ancestors u_1 of u'_1 ; conversely, for deleting an edge (u'_1, u'_2) we remove u'_1 from $S(u_2)$ and u'_2 from $S(u_1)$. Since there are at most $2D$ ancestors, this yields the following lemma:

Lemma 4. `newEdge` (u_1, u_2) . . . `deleteEdge` (u_1, u_2) . . . $\mathcal{O}(D \log n)$. . .

For deleting a leaf u , we first delete all incident edges with `deleteEdge`, which implicitly updates all affected sets $S(\cdot)$. Next, we update the order maintenance and the level ancestor data structures and remove the leaf from its tree. At each ancestor u' of u we possibly have to update the values $\min(u')$ and $\max(u')$ to u 's predecessor or successor in the ordered list of tree nodes.

When inserting a new leaf u' as a child of node u , we first insert u' right before $\max(u)$ into the order maintenance structure. Then we add u' to the level ancestor data structure and insert it into the tree as a child of u . If u was an inner node before this operation, the values $\min(u)$ and $\max(u)$ remain correct. But if u was a leaf, i. e., $\max(u) = \min(u)$, we have to set $\min(u) = u'$; this may cause further updates of the $\min(\cdot)$ values at ancestors of u . Since insertion and deletion in the order maintenance as well as the level ancestor data structure take constant time, this yields the following lemma:

Lemma 5. . . . $\mathcal{O}(D)$. . .

2.2 Higher Dimensions

The data structure described so far maintains a dynamic set of pairs $(u_1, u_2) \in V(T_1) \times V(T_2)$, while both trees are dynamic in regard to insertion and deletion of leaves. It provides the retrieval operations `edgeQuery`, `edgeReport`, and `edgeExpand`. We will give a recursive description for the higher dimensional data structure with the two-dimensional case as basis.

Suppose that there is already such a data structure for the case d , i. e., for maintaining hyperedges between nodes of d trees. The $(d + 1)$ -dimensional data structure stores at each node $u_1 \in V(T_1)$ the set

$$S_{d+1}(u_1) = \{(u'_1, u'_2, \dots, u'_{d+1}) \in E(G) \mid u'_1 \in \text{desc}(u_1)\},$$

i. e., all hyperedges incident with descendants of u_1 . Disregarding the first element of the hyperedges, we use a separate d -dimensional data structure for each set $S_{d+1}(\cdot)$. In other words, we store the $(d + 1)$ -dimensional hyperedges in a d -dimensional data structure according to their projections onto the last d elements.

We can implement `edgeQuery`(u_1, u_2, \dots, u_{d+1}) as `edgeQuery`(u_2, \dots, u_{d+1}) on the d -dimensional data structure stored at u_1 . An `edgeReport` query is forwarded similarly; the edges it returns are already the correct $(d+1)$ -dimensional result, because the d -dimensional data structure contains the original hyperedges. The operation `edgeExpand`((u_1, u_2, \dots, u_{d+1}), j) for $j \neq 1$ is implemented as an `edgeExpand`((u_2, \dots, u_{d+1}), j) on the d -dimensional data structure at the node u_1 . For expanding a hyperedge at its first element ($j = 1$), we build the same data structure designating some other tree to be T_1 , for instance T_2 .

Theorem 1. $\mathcal{O}(m(2D)^{d-1})$ \dots
 $d \dots$

For $d = 2$, all bounds in Table 1 follow directly from Lemmas 1, 2, 3, 4, and 5.

For $d > 2$, an edge (u_1, \dots, u_d) contributes an entry to the lower dimensional data structure at each ancestor of u_1 and at each ancestor of u_2 (assuming that T_2 is the tree designated to be T_1 for the second data structure). These are $\mathcal{O}(2D)$ entries; by induction each entry uses $\mathcal{O}((2D)^{(d-1)-1})$ space in the lower dimensional data structure, which gives a total of $\mathcal{O}(m(2D)^{d-1})$ space. Inserting or deleting an edge is implemented as one insert or delete operation in a lower dimensional data structure for each ancestor in the two dimensions, each of which takes $\mathcal{O}((2D)^{(d-1)-1} \log n)$ by induction. All retrieval operations `edgeQuery`, `edgeReport`, and `edgeExpand` are forwarded to an appropriate lower dimensional data structure; this recursion ends at some two dimensional data structure, where the operation is implemented as described in Sect. 2.1. Hence, we get additional $d - 1$ steps for the recursion. Inserting and deleting a leaf is exactly the same as in the two dimensional case. \square

In [3] \dots [14] are used to improve the space bound. For a tree T , an edge $(\text{parent}(u), u)$ is \dots if $2|\text{desc}(u)| \leq |\text{desc}(\text{parent}(u))|$ and \dots otherwise. The compressed tree $C(T)$ evolves from T by contracting all heavy paths into their respective topmost node. This technique could be employed here as well, but maintaining $C(T)$ subject to insertion and deletion of leaves into the original tree T is not straightforward. The problem is that these modifications can change the status of tree edges at ancestors of the affected node from light to heavy and vice versa. In the compressed tree this results in adding or removing an inner node. Especially for a new inner node this is expensive, because we have to equip the new node with appropriate data structures, e. g., the set $S(\cdot)$. While reducing the space bound, using compressed trees increases the time bounds of most operations by a factor of $\mathcal{O}(\log n / (\log \log n)^2)$. In [3] the trees are stratified recursively to improve these time bounds again. Unfortunately, the stratification arguments are faulty [15]; therefore, only the results without stratification are listed in Table 1.

3 Maintaining Hierarchical Graph Views

A compound graph $\Gamma = (V, E_i, E_a)$ [4] consists of nodes V , inclusion edges E_i , and adjacency edges E_a . It is required that the inclusion digraph $T = (V, E_i)$ is a tree and no adjacency edge connects a node to one of its descendants or ancestors; see Figs. 2 and 3. A view U is a graph with nodes $V(U) \subseteq V$ such that $\forall u, v \in V(U) : \text{desc}(u) \cap \text{desc}(v) = \emptyset$, i.e., the nodes of the view are not related in terms of the inclusion tree. Two nodes $u, v \in V(U)$ are connected by an adjacency edge if and only if there are nodes $u' \in \text{desc}(u)$ and $v' \in \text{desc}(v)$ such that u' and v' are connected by an adjacency edge $(u', v') \in E_a$; see Fig. 4.

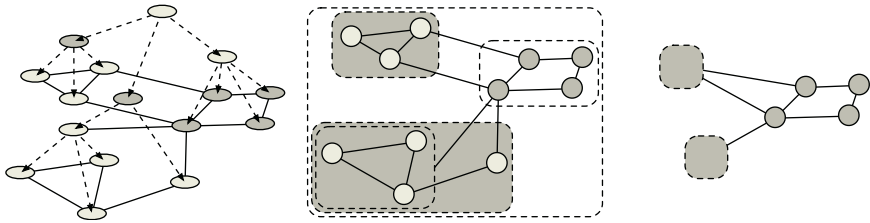


Fig. 2. An example of a compound graph: the directed edges form the inclusion digraph T ; the undirected ones are the adjacency edges E_a . **Fig. 3.** The same compound graph as in Fig. 2, but T is depicted by the dashed rectangles. **Fig. 4.** The view consisting of the darker shaded nodes of the compound graph in Figs. 2 and 3.

Given a compound graph Γ and a view U , the dynamic view problem, according to [2], is to efficiently perform the following operations on U :

- **expand**(v), where $v \in V(U)$; replaces node v with its children, i.e., the result is the view U' with nodes $V(U') = V(U) \setminus \{v\} \cup \text{children}(v)$,
- **contract**(v), where $\text{children}(v) \subseteq V(U)$; contracts all children of v , i.e., the result is the view U' with nodes $V(U') = V(U) \setminus \text{children}(v) \cup \{v\}$.

In the new dynamic leaves variant of this problem, the compound graph Γ is subject to the following modifications:

- **newEdge**(u, v), where $u, v \in V$, $u \notin \text{desc}(v)$, and $v \notin \text{desc}(u)$; adds a new adjacency edge (u, v) to Γ ,
- **deleteEdge**(u, v), where $(u, v) \in E_a$; removes adjacency edge (u, v) from Γ ,
- **newLeaf**(u), where $u \in V$; adds a new node v to Γ and a new inclusion edge (u, v) , i.e., v becomes a child of u in the inclusion tree,
- **deleteLeaf**(u), where u is a leaf in the inclusion tree; removes u from G .

Besides compound graphs, there are other concepts for extending graphs with a hierarchical structure [5, 16, 17, 18], among which *tree cross products* [5] are very popular; they consist of a base graph and a tree whose leaves are exactly the

Table 2. Results and comparison for the graph view maintenance problem. Let $D = \text{depth}(T)$, $s = \min\{D, \log n\}$, and $\text{Opt}(U, v) = \sum_{v' \in \text{children}(v)} |\text{adj}_U(v')|$. For **expand**(v), U' denotes the view after expanding v in U . The bounds labeled with ^{exp} are expected, all others are worst-case

	Approach of [2]	Approach of [3]	Our data structure
Space	$\mathcal{O}(ms^2)$	$\mathcal{O}(mD \log \log n)$	$\mathcal{O}(mD)$
expand (v)	$\mathcal{O}(\text{Opt}(U', v))$	$\mathcal{O}(\text{Opt}(U', v) \log \log n)$	$\mathcal{O}(\text{Opt}(U', v) \log n)$
contract (v)	$\mathcal{O}(\text{Opt}(U, v))$	$\mathcal{O}(\text{Opt}(U, v))$	$\mathcal{O}(\text{Opt}(U, v))$
newEdge (u, v)	$\mathcal{O}^{\text{exp}}(s^2 \log n)$	$\mathcal{O}(D \log \log n)$	$\mathcal{O}(D \log n)$
deleteEdge (u, v)	$\mathcal{O}^{\text{exp}}(s^2 \log n)$	$\mathcal{O}(D \log \log n)$	$\mathcal{O}(D \log n)$
newLeaf (u)	n/a	n/a	$\mathcal{O}(D)$
deleteLeaf (u)	n/a	n/a	$\mathcal{O}(D)$

nodes of the base graph. Consequently, clustered graphs have adjacency edges only between leaves of the inclusion tree, whereas \dots allow them between any pair of tree nodes such that neither is a descendant of the other. In [2,3] data structures for maintaining views of clustered graphs under **expand** and **contract** operations are described; these are either static or allow insertion and deletion of adjacency edges. Efficient data structures that additionally support modifications of the node set were left as an open problem [2]. Providing insertion and deletion of leaves, we solve this problem partially; see also [19] for a detailed description on directly applying the ideas of Sect. 2.1 to this problem. Table 2 summarizes our results and compares them to the other approaches.

The graph view maintenance problem for compound graphs can be reduced to a two-dimensional tree cross product; see [3, Sect. 5.1]. We set $T_1 = T_2 = T$ and interpret an adjacency edge $(u, v) \in E_a$ as an edge connecting $u \in T_1$ and $v \in T_2$. Clearly, determining whether there is an induced edge between two nodes u and v becomes an **edgeQuery**; **newEdge** and **deleteEdge** directly map to corresponding operations for tree cross products. Inserting or deleting leaves in the inclusion tree engenders a **newLeaf** or **deleteLeaf** operation on both T_1 and T_2 .

For expanding a view U at the node v , we use an **expandEdge**($(v, w), 1$) for each edge (v, w) incident to v in the view U ; this determines all the children of v inheriting the edge (v, w) . Contracting a view at the node v is straightforward: all the children of v are removed and v is connected to all former neighbors of children of v .

Theorem 2. $\dots D = \text{depth}(T) \dots \mathcal{O}(mD) \dots$

As in [2, 3], let $\text{Opt}(U, v) = \sum_{v' \in \text{children}(v)} |\text{adj}_U(v')|$, where $\text{adj}_U(v')$ are the edges incident to v' in the view U . The number of items that have to be added or removed during $\text{expand}(v)$ is bounded by $\mathcal{O}(\text{Opt}(U', v))$, where U' is the view after expanding v in U . Similarly, the number of items affected by a $\text{contract}(v)$ is bounded by $\mathcal{O}(\text{Opt}(U, v))$. By traversing all edges incident to children of v , we can find the neighbors of v in U' , where U' is the view resulting from contracting the children of v in U . Hence, $\text{contract}(v)$ takes $\mathcal{O}(\text{Opt}(U, v))$ time. $\text{expand}(v)$ is bounded by $\mathcal{O}(\text{Opt}(U', v) \log n)$, for we have to expand each edge incident to v in U ; see Algorithm 1. Note that this does not yield the edges between two children of v : we maintain these edges separately in a list at every node of the tree. Since each edge is stored exactly once in such a list, namely at the nearest common ancestor of its end nodes, this uses $\mathcal{O}(m)$ additional space, which does not violate the $\mathcal{O}(mD)$ space bound. Clearly, it takes additional $\mathcal{O}(D)$ time for updating these lists, which is possible within the $\mathcal{O}(D \log n)$ bound for inserting and deleting edges. All other bounds follow immediately from Theorem 1. \square

4 Conclusion

We have presented an efficient data structure for range searching over tree cross products, where the trees are dynamic with regard to insertion and deletion of leaves. As summarized in Table 1, our approach can compete well with the one it extends [3]. So far, it is the only data structure for tree cross products where the node set is dynamic. Applying it to graph view maintenance, we have partially solved the dynamic graph and tree variant, an open problem in [2]. The comparison in Table 2 shows that our solution matches with the more static ones, but additionally provides insertion and deletion of graph nodes. A data structure for the dynamic graph and tree variant, i. e., with splitting and merging of clusters, remains an open problem.

Acknowledgments

I would like to thank Adam Buchsbaum for the enlightening discussions on the details of [3] and for his valuable comments on drafts of this paper. Furthermore, I am grateful to Franz Brandenburg, Christian Bachmaier, and Falk Schreiber for their suggestions.

References

1. Brandenburg, F.J., Forster, M., Pick, A., Raitner, M., Schreiber, F.: Biopath-exploration and visualization of biochemical pathways. In Mutzel, P., Jünger, M., eds.: Graph Drawing Software. Mathematics and Visualization. Springer (2003) 215–236
2. Buchsbaum, A.L., Westbrook, J.R.: Maintaining hierarchical graph views. In: Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA). (2000) 566–575

3. Buchsbaum, A.L., Goodrich, M.T., Westbrook, J.R.: Range searching over tree cross products. In Paterson, M., ed.: Proc. 8th European Symposium on Algorithms (ESA). Volume 1879 of LNCS. (2000) 120–131
4. Sugiyama, K., Misue, K.: Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics* **21** (1991) 876–892
5. Feng, Q.W., Cohen, R.F., Eades, P.: How to draw a planar clustered graph. In Du, D.Z., Li, M., eds.: Proc. 1st Intl. Conference on Computing and Combinatorics (COCOON). Volume 959 of LNCS. (1995) 21–30
6. Bender, M.A., Richard, C., Demaine, E.D., Farach-Colton, M., Zito, J.: Two simplified algorithms for maintaining order in a list. In Möhring, R.H., Raman, R., eds.: Proc. 10th European Symposium on Algorithms (ESA). Volume 2461 of LNCS. (2002) 152–164
7. Dietz, P.F., Sleator, D.D.: Two algorithms for maintaining order in a list. In: 9th ACM Symposium on Theory of Computing (STOC). (1987) 365–372
8. Willard, D.E.: Good worst-case algorithms for inserting and deleting records in dense sequential files. In: Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data. (1986) 251–260
9. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. *Mathematical Systems Theory* **10** (1977) 99–127
10. Berkman, O., Vishkin, U.: Finding level ancestors in trees. *Journal of Computer and System Sciences* **48** (1994) 214–230
11. Bender, M.A., Farach-Colton, M.: The level ancestor problem simplified. In Rajsbaum, S., ed.: Proc. 5th Latin American Symposium on Theoretical Informatics (LATIN). Volume 2286 of LNCS. (2002) 508–515
12. Alstrup, S., Holm, J.: Improved algorithms for finding level ancestors in dynamic trees. In Montanari, U., Rolim, J.D.P., Welzl, E., eds.: Automata, Languages and Programming, 27th International Colloquium, ICALP 2000. Volume 1853 of LNCS. (2000) 73–84
13. Dietz, P.F.: Finding level ancestors in dynamic trees. In Dehne, F.K.H.A., Sack, J.R., Santoro, N., eds.: Algorithms and Data Structures, 2nd Workshop WADS '91. Volume 519 of LNCS. (1991) 32–40
14. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing* **13** (1984) 338–355
15. Buchsbaum, A.L.: Personal communication
16. Harel, D.: On visual formalisms. *Comm. of the ACM* **31** (1988) 588–600
17. Lai, W., Eades, P.: A graph model which supports flexible layout functions. Technical Report 96–15, University of Newcastle (1996)
18. Raitner, M.: HGV: A library for hierarchies, graphs, and views. In Goodrich, M.T., Kobourov, S.G., eds.: Proc. 10th Intl. Symposium on Graph Drawing (GD). Volume 1528 of LNCS. (2002) 236–243
19. Raitner, M.: Maintaining hierarchical graph views for dynamic graphs. Technical Report MIP-0403, Universität Passau (2004)

Spanners, Weak Spanners, and Power Spanners for Wireless Networks

Christian Schindelhauer*, Klaus Volbert*, and Martin Ziegler*

Heinz Nixdorf Institute, Paderborn University
Institute of Computer Science,
{schindel, kvolbert, ziegler}@uni-paderborn.de

Abstract. For $c \in \mathbb{R}$, a c -spanner is a subgraph of a complete Euclidean graph satisfying that between any two vertices there exists a path of weighted length at most c times their geometric distance. Based on this property to approximate a complete weighted graph, sparse spanners have found many applications, e.g., in FPTAS, geometric searching, and radio networks. In a *weak* c -spanner, this path may be arbitrary long but must remain within a disk of radius c -times the Euclidean distance between the vertices. Finally in a c -power spanner, the total energy consumed on such a path, where the energy is given by the sum of the *squares* of the edge lengths on this path, must be at most c -times the square of the geometric distance of the direct link.

While it is known that any c -spanner is also both a weak C_1 -spanner and a C_2 -power spanner (for appropriate C_1, C_2 depending only on c but not on the graph under consideration), we show that the converse fails: There exists a family of c_1 -power spanners that are no weak C -spanners and also a family of weak c_2 -spanners that are no C -spanners for any fixed C (and thus no uniform spanners, either). However the deepest result of the present work reveals that any weak spanner *is* also a uniform power spanner. We further generalize the latter notion by considering (c, δ) -power spanners where the sum of the δ -th powers of the lengths has to be bounded; so $(\cdot, 2)$ -power spanners coincide with the usual power spanners and $(\cdot, 1)$ -power spanners are classical spanners. Interestingly, these (\cdot, δ) -power spanners form a strict hierarchy where the above results still hold for any $\delta \geq 2$; some even hold for $\delta > 1$ while counterexamples exist for $\delta < 2$. We show that every self-similar curve of fractal dimension $d > \delta$ is no (C, δ) -power spanner for any fixed C , in general.

1 Motivation

Spanners have appeared in Computer Science with the advent of Computational Geometry [4, 18], raised further in interest as a tool for approximating NP-hard problems [13] and, quite recently, for routing and topology control in ad-hoc networks [1, 12, 8, 7, 11]. Roughly speaking, they approximate the complete Euclidean graph on a set of geometric vertices while having only linearly many edges. The formal condition for a c -spanner

* Partially supported by the DFG-Sonderforschungsbereich 376 and by the EU within 6th Framework Programme under contract 001907 “Dynamically Evolving, Large Scale Information Systems” (DELIS).

$G = (V, E)$ is that between any two $u, v \in V$, the edge (u, v) may be absent provided there exists a path in G from u to v of length at most c -times the Euclidean distance between u and v ; see Figure 1(a). In particular, this path remains within a circle around u of radius c . For applications in geometric searching [5], it has turned out that graphs with the latter, weaker condition suffice; see Figure 1(b). In [10] such weak spanners are used to approximate congestion-optimal radio networks. Several constructions yield both spanners [6] and weak spanners [5] with arbitrarily describable approximation ratio. Among them, some furthermore benefit from nice locality properties which led to successful applications in ad-hoc routing networks [7, 9, 17, 16]. However in order to restrict the power consumption during such a communication (which for physical reasons grows quadratically with the Euclidean length of each link), one is interested in routing paths, say between u and v , whose sum of *squares* of lengths of the individual steps is bounded by c -times the square of the Euclidean distance between u and v ; see Figure 1(c). Such graphs are known as c -power spanners [7, 9].

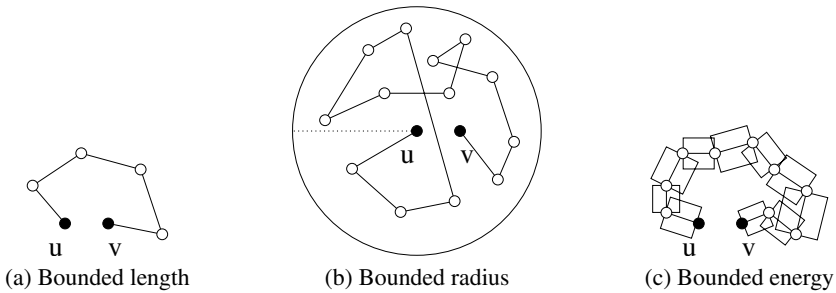


Fig. 1. Spanner, Weak Spanner and Power Spanner

Finally, when power consumption is of minor interest but the routing time is dominated by the number of individual steps, sparse graphs are desired which between any vertices u and v provides a path containing at most c further vertices. These are the so-called c -hop spanners [1]. In this paper, we investigate the relations between these various types of spanners. Observe that any strongly connected finite geometric graph is a C -spanner for *some*¹ value C . Therefore the question on the relation between spanners and weak spanners rather asks whether any weak c -spanner is a C -spanner for some value C depending *only* on c . Based on a construction from [3], we answer this to the negative. For some weak c -spanners it is proved that they are also C -power spanners for some value C [7, 8] using involved constructions. One major contribution of our work generalizes and simplifies such results by showing that in the plane in fact *any* weak c -spanner is a C -power spanner with $C = O(c^8)$. Moreover, we investigate the notion of a (c, δ) -power spanner [7] which

¹ Consider for any pair u, v of vertices some path from u to v and the ratio of its length to the distance between u and v ; then taking for C the maximum over the (finitely many) pairs u, v will do.

- for $\delta = 1$ coincides with c -spanners
- for $\delta = 2$ coincides with (usual) c -power spanners
- for $\delta = 0$ coincides with c -hop spanners, i.e. graphs with diameter c
- for $\delta > 2$ reflects transmission properties of radio networks (e.g., for δ up to 6 or even 8).

We show that these form a strict hierarchy: For $\Delta > \delta > 0$, any (c, δ) -power spanner is also a (C, Δ) -power spanner with C depending only on c and Δ/δ ; whereas we give examples of (C, Δ) -power spanners that are no (c, δ) -power spanner for any fixed c . Our main contribution is that any weak c -spanner is also a (C, δ) -power spanner for arbitrary $\delta \geq 2$ with C depending on c and δ only. We finally show that this claim is best possible by presenting, for arbitrary $\delta < 2$, weak c -spanners which are no (C, δ) -power spanner for any fixed C .

This paper is organized as follows: Section 2 formally defines the different types of spanners under consideration. Section 3 shows that, while any c -spanner is also a weak c -spanner, a weak c -spanner is in general no C -spanner for any C depending just on c . Section 4 similarly reveals the relations between spanners and power spanners. The central Section 5 of the this work investigates the relation between weak spanners and power spanners. Theorem 3 gives an example of a power spanner which is no weak spanner. Our major contributions then prove that, surprisingly, any weak c -spanner is also a C -power spanner with C depending only on c . For different values of δ , we obtain different upper bounds to C in terms of c : For $\delta = 2$ (power spanners in the original sense), we show $C \leq \mathcal{O}(c^8)$, see Theorem 5; for $\delta > 2$, we have $C \leq \mathcal{O}(c^{2+\delta}/(1 - 2^{2-\delta}))$, see Theorem 4. However for $\delta < 2$, we present counterexamples of unbounded C , that is, in this case provably *not* any weak c -spanner is a (C, δ) -power spanner. Further, we generalize our construction and analysis to self-similar fractal curves. Section 6 finally shows that for different δ , the respective classes of (\cdot, δ) -power spanners form a strict hierarchy. In Section 7 we discuss extensions of our results to higher-dimensional cases, before we conclude this work presenting applications of our results concerning power-efficient wireless networks in Section 8.

2 Preliminaries

We focus on the two-dimensional case, that is, directed graphs $G = (V, E)$ with finite $V \subseteq \mathbb{R}^2$; extensions to higher dimensions are discussed in Section 7. Let $|\mathbf{u} - \mathbf{v}|$ denote the Euclidean distance between $\mathbf{u}, \mathbf{v} \in V$. A *path* from \mathbf{u} to \mathbf{v} in G is a finite sequence $P = (\mathbf{u} = \mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_\ell = \mathbf{v})$ of vertices $\mathbf{u}_i \in V$ such that $(\mathbf{u}_{i-1}, \mathbf{u}_i) \in E$ for all $i = 2, \dots, \ell$. Occasionally, we also encounter the more general situation of a path from \mathbf{u} to \mathbf{v} not necessarily in G ; this means that $\mathbf{u}_i \in V$ still holds but the requirement $(\mathbf{u}_i, \mathbf{u}_{i+1}) \in E$ is dropped. The *radius* of a path P is the real number $\max_{i=2}^{\ell} |\mathbf{u} - \mathbf{u}_i|$. The (Euclidean) *length* of P is given by $\sum_{i=2}^{\ell} |\mathbf{u}_i - \mathbf{u}_{i-1}|$; the *hop length* is $\ell - 1$; for $\delta \geq 0$,

$$\|P\|^\delta := \sum_{i=2}^{\ell} |\mathbf{u}_i - \mathbf{u}_{i-1}|^\delta \quad (1)$$

denotes the δ -cost of P . The length is just the 1-cost whereas the hop length coincides with the 0-cost.

Definition 1. Let $G = (V, E)$ be a geometric directed graph with finite $V \subseteq \mathbb{R}^2$ and $c > 0$. G is a c -spanner, if for all $\mathbf{u}, \mathbf{v} \in V$ there is a path P from \mathbf{u} to \mathbf{v} in G of length $\|P\|^1$ at most $c \cdot |\mathbf{u} - \mathbf{v}|$. G is a weak c -spanner, if for all $\mathbf{u}, \mathbf{v} \in V$ there is a path P from \mathbf{u} to \mathbf{v} in G of radius at most $c \cdot |\mathbf{u} - \mathbf{v}|$. For $\delta \geq 0$, G is a (c, δ) -power spanner if for all $\mathbf{u}, \mathbf{v} \in V$ there is a path P from \mathbf{u} to \mathbf{v} in G of δ -cost $\|P\|^\delta$ at most $c \cdot |\mathbf{u} - \mathbf{v}|^\delta$. G is a c -power spanner, if G is a $(c, 2)$ -power spanner. The factor c is called length stretch factor, weak stretch factor or power stretch factor, respectively.

Informally (see Figure 1), in a c -spanner there exists between two arbitrary vertices a path of length at most c -times the Euclidean distance between these vertices (bounded length). In a weak c -spanner, this path may be arbitrary long but must remain within a disk of radius c -times the Euclidean distance between the vertices (bounded radius). Finally in a c -power spanner, the energy consumed on such a path (e.g., the sum of the squares of the lengths of its constituting edges) must be at most c -times the one consumed on a putative direct link (bounded cost). Sometimes we shorten the notion of spanner, weak spanner and power spanner and omit constant parameters. So, if we say that a family of graphs is a *spanner*, then there exists a constant c such that all its members are c -spanners.

The attentive reader might have observed that our Definition 1 does not exactly match that from [7]. The latter required that the 2-cost of some path P from \mathbf{u} to \mathbf{v} in G is bounded by c -times the 2-cost of *any* path Q (not necessarily in G) from \mathbf{u} to \mathbf{v} . However, both approaches are in fact equivalent:

Lemma 1. Let $G = (V, E)$ be a (c, δ) -power spanner, $\mathbf{u}, \mathbf{v} \in V$, and let Q denote some path Q (not necessarily in G) from \mathbf{u} to \mathbf{v} of minimum δ -cost. Then there is a path P in G from \mathbf{u} to \mathbf{v} of δ -cost $\|P\|^\delta$ at most $c \cdot \|Q\|^\delta$.

Proof. Let $Q = (\mathbf{u} = \mathbf{q}_1, \dots, \mathbf{q}_L = \mathbf{v})$. For each $i = 2, \dots, L$ there exists by presumption a path P_i in G from \mathbf{q}_{i-1} to \mathbf{q}_i of δ -cost at most $c \cdot |\mathbf{q}_i - \mathbf{q}_{i-1}|^\delta$. The concatenation of all these paths yields a path P from \mathbf{u} to \mathbf{v} in G with δ -cost $\|P\|^\delta$ at most $c \cdot \|Q\|^\delta$.

3 Spanners Versus Weak Spanners

Every c -spanner is also a weak c -spanner. Our first result shows that the converse fails in general.



Fig. 2. Construction of EPPSTEIN provably yields no spanner but a weak spanner

Theorem 1. There is a family of graphs $G = (V, E)$ with $V \subseteq \mathbb{R}^2$ all of which are weak $(\sqrt{3} + 1/2)$ -spanners but no C -spanners for any fixed $C \in \mathbb{R}$.

Proof. We show the claim using the fractal construction presented in [3] (see Figure 2). We briefly review its recursive definition which is similar to that of a KOCH Curve. At the beginning there are two vertices with distance 1. In the following steps we replace each edge by 5 new edges of equal length as follows: one horizontal, one at angle $\pi/4$, a second horizontal, another one at angle $-\pi/4$ and a third horizontal. After i steps we have a graph consisting of 5^i edges and $5^i + 1$ vertices. As shown in [3] this graph has unbounded length stretch factor. We argue that there exists a constant c such that it is a weak c -spanner. It is known that the area under the constructed curve is bounded by a constant and that the path between two vertices $u, v \in V$ lies completely in a disk around the midpoint of the segment between u and v with radius at most $(2 \cdot \sqrt{3}/2) = \sqrt{3}$ (see KOCH's Snowflake, Figure 6). Applying Observation 2 proves the claim.

The following observation says that, up to constants, it makes no difference in the definition of a weak spanner whether the radius is bounded with respect to center \mathbf{u} (the starting one of the two points) or with respect to center $(\mathbf{u} + \mathbf{v})/2$ (the midpoint of the segment between the two points).

Observation 2. *Let $P = (\mathbf{u} = \mathbf{u}_1, \dots, \mathbf{u}_\ell = \mathbf{v})$ be a path in the geometric graph $G = (V, E)$ such that $|\mathbf{u} - \mathbf{u}_i| \leq c \cdot |\mathbf{u} - \mathbf{v}|$ for all $i = 1, \dots, \ell$. Then, $\mathbf{w} := (\mathbf{u} + \mathbf{v})/2$ satisfies by the triangle inequality*

$$|\mathbf{w} - \mathbf{u}_i| = |\mathbf{u} - \mathbf{u}_i + (\mathbf{v} - \mathbf{u})/2| \leq |\mathbf{u} - \mathbf{u}_i| + |\mathbf{v} - \mathbf{u}|/2 \leq (c + \frac{1}{2}) \cdot |\mathbf{u} - \mathbf{v}| .$$

Conversely if P has $|\mathbf{w} - \mathbf{u}_i| \leq c \cdot |\mathbf{u} - \mathbf{v}|$ for all i , then

$$|\mathbf{u} - \mathbf{u}_i| = |\mathbf{w} - \mathbf{u}_i + (\mathbf{u} - \mathbf{v})/2| \leq |\mathbf{w} - \mathbf{u}_i| + |\mathbf{u} - \mathbf{v}|/2 \leq (c + \frac{1}{2}) \cdot |\mathbf{u} - \mathbf{v}| .$$

4 Spanners Versus Power Spanners

In [7] it is shown that, for $\delta > 1$, every c -spanner is also a (c^δ, δ) -power spanner. However, conversely, for any $\delta > 1$, there are (c, δ) -power spanners which are no C -spanners for any fixed C : This follows from Theorem 3 as any C -spanner is a weak C -spanner as well.

5 Weak Spanners Versus Power Spanners

Now, we turn to the main contribution of the present paper and present our results concerning the relation between weak spanners and power spanners. Surprisingly, it turns out that any weak c -spanner is also a C -power spanner for some C depending only on c . But first observe that the converse in general fails:

Theorem 3. *In the plane and for any $\delta > 1$, there is a family of (c, δ) -power spanners which are no weak C -spanners for any fixed C .*

Proof. Let $V := \{\mathbf{u} = \mathbf{v}_1, \dots, \mathbf{v} = \mathbf{v}_n\}$ be a set of n vertices placed on a circle scaled such that the Euclidean distance between \mathbf{u} and \mathbf{v} is 1 and $|\mathbf{v}_i - \mathbf{v}_{i+1}| = 1/i$ for all

$i = 1, \dots, n - 1$. Now, consider the graph $G = (V, E)$ with edges $(\mathbf{v}_i, \mathbf{v}_{i+1})$. First observe that G is a (c, δ) -power spanner with c independent of n . Indeed, its δ -power stretch factor is dominated by the δ -cost of the (unique) path P in G from \mathbf{u} to \mathbf{v} which amounts to

$$\|P\|^\delta = \sum_{i=1}^{n-1} (1/i)^\delta \leq \sum_{i=1}^{\infty} (1/i)^\delta =: c$$

a convergent series since $\delta > 1$. This is compared to the cost of the direct link from \mathbf{u} to \mathbf{v} of 1. On the other hand, the Euclidean length (that is, the 1-cost) of the polygonal chain from \mathbf{u} to \mathbf{v} is given by the unbounded harmonic series $\sum_{i=1}^{n-1} (1/i) = \Theta(\log n)$. Therefore also the radius of this polygonal chain cannot be bounded by any C independent of n , either.

In the sequel, we show that, conversely, any weak c -spanner is a (C, δ) -power spanner for both $\delta > 2$ (Section 5.1) and $\delta = 2$ (Section 5.2) with C depending only on c and δ . A counter-example in Section 5.3 reveals that this however does not hold for $\delta < 2$.

5.1 Weak Spanner Implies Power Spanner for $\delta > 2$

In this subsection, we show that any weak c -spanner is also a (C, δ) -power spanner for any $\delta > 2$ with C depending only on c and δ . By its definition, between vertices \mathbf{u}, \mathbf{v} , there exists a path P in G from \mathbf{u} to \mathbf{v} that remains within a disk around \mathbf{u} of radius $c \cdot |\mathbf{u} - \mathbf{v}|$. However on the course of this path, two of its vertices \mathbf{u}' and \mathbf{v}' might come very close so that P , considered as a subgraph of G , in general is no weak c -spanner. On the other hand, G being a weak c -spanner, there exists also between \mathbf{u}' and \mathbf{v}' a path P' of small radius. Based on such repeated applications of the weak spanner property, we first assert the existence of a path which, considered as a subgraph of G , is weak $2c$ -spanner.

Definition 2. Let $G = (V, E)$ be a directed geometric graph and $e_1 := (\mathbf{u}_1, \mathbf{v}_1)$, $e_2 := (\mathbf{u}_2, \mathbf{v}_2)$ two of its edges. By their distance we mean the number

$$\min \{ |\mathbf{u}_1 - \mathbf{u}_2|, |\mathbf{v}_1 - \mathbf{v}_2|, |\mathbf{u}_1 - \mathbf{v}_2|, |\mathbf{v}_1 - \mathbf{u}_2| \} ,$$

that is, the Euclidean distance of a closest pair of their vertices (see Figure 3(b)).

Lemma 2. Let $G = (V, E)$ be a weak c -spanner and $\mathbf{u}, \mathbf{v} \in V$. Then there is a path P from \mathbf{u} to \mathbf{v} in G which, as a subgraph of G , is a weak $2c$ -spanner.

Proof. The idea is to take the path P asserted by the weak spanner property for \mathbf{u} and \mathbf{v} and to, for any pair \mathbf{u}', \mathbf{v}' of vertices on P for which P violates that property, locally replace that part of P by a path from \mathbf{u}' to \mathbf{v}' in G . However for these iterated improvements to eventually terminate, we perform them in decreasing order of the lengths of the edges involved.

W.l.o.g. we assume $|\mathbf{u} - \mathbf{v}| = 1$. Since G is a weak c -spanner there exists a path $P = (\mathbf{u} = \mathbf{u}_1, \dots, \mathbf{u}_\ell = \mathbf{v})$ from \mathbf{u} to \mathbf{v} in G that lies completely within a disk

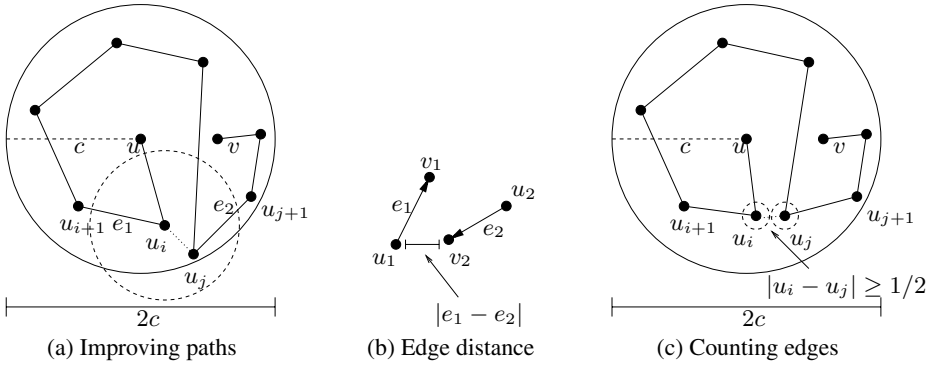


Fig. 3. Construction and analysis of a path with low power stretch in a weak spanner

around \mathbf{u} of radius c . In particular, any edge on this path has a length of at most $2c$, see Figure 3(a).

Now, consider all edges on this path of length between c and $2c$. For any pair $e_1 = (\mathbf{u}_i, \mathbf{u}_{i+1})$ and $e_2 = (\mathbf{u}_j, \mathbf{u}_{j+1})$ with $j > i$ closer than $\frac{1}{2}$ (Definition 2), improve that part of P by replacing it with a path according to the weak c -stretch property. Observe that, since the improvement is applied to vertices of distance at most $\frac{1}{2}$, this sub-path remains within a disk of radius $c/2$; in particular any edge introduced to P has length at most c and thus does not affect the edges of length between c and $2c$ currently considered. Moreover, after having performed such improvements to all edges of length between c and $2c$, the resulting path P' , although it might now leave the disk around \mathbf{u} of radius c , it does remain within radius $c + c/2$.

Next, we apply the same process to edges of length between c and $c/2$ and perform improvements on those closer than $\frac{1}{4}$. The thus obtained path P'' remains within a disk of radius $c + c/2 + c/4$ while, for any pair of vertices \mathbf{u}' and \mathbf{v}' improved in the previous phase, the sub-path between them might increase in radius from $c \cdot |\mathbf{u}' - \mathbf{v}'|$ to at most $(c + c/2) \cdot |\mathbf{u}' - \mathbf{v}'|$.

As G is a finite graph, repeating this process for edges of length between $c/2$ and $c/4$ and so on, will eventually terminate and yield a path \tilde{P} from \mathbf{u} to \mathbf{v} remaining within a disk of radius $c + c/2 + c/4 + \dots = 2c$. Moreover, for *any* pair of vertices \mathbf{u}' , \mathbf{v}' in P , the sub-path between them has radius at most $(c + c/2 + c/4 + \dots) \cdot |\mathbf{u}' - \mathbf{v}'|$ which proves that \tilde{P} is indeed a weak $2c$ -spanner.

Lemma 3. *Let $P = (\mathbf{u}_1, \dots, \mathbf{u}_\ell)$ be a weak $2c$ -spanner, $\mathbf{u}_i \in \mathbb{R}^2$, $|\mathbf{u}_1 - \mathbf{u}_\ell| = 1$. Then, P contains at most $(8c + 1)^2$ edges of length greater than c ; more generally, P contains at most $(8c + 1)^2 \cdot 4^k$ edges of length greater than $c/2^k$.*

Proof. Consider two edges $(\mathbf{u}_i, \mathbf{u}_{i+1})$ and $(\mathbf{u}_j, \mathbf{u}_{j+1})$ on P both of length at least c with $j > i$. P being a $2c$ -weak spanner implies that, between vertices \mathbf{u}_i and \mathbf{u}_j , the sub-path in P from \mathbf{u}_i to \mathbf{u}_j (which is unique and passes through \mathbf{u}_{i+1}), satisfies $c \leq |\mathbf{u}_i - \mathbf{u}_{i+1}| \leq 2c \cdot |\mathbf{u}_i - \mathbf{u}_j|$; hence, $|\mathbf{u}_i - \mathbf{u}_j| \geq \frac{1}{2}$, see Figure 3(c). In particular, placing an Euclidean disk B_i of radius $\frac{1}{4}$ around each starting vertex \mathbf{u}_i of an edge of

length at least c results in these disks being mutually disjoint. If m denotes the number of edges of length at least c , these disks thus cover a total area of $m\pi(\frac{1}{4})^2$. On the other hand, as all \mathbf{u}_i lie within a single disk around \mathbf{u}_1 of radius $2c$, all disks B_i together cover an area of at most $\pi(2c + \frac{1}{4})^2$. Therefore,

$$m \leq \frac{\pi(2c + \frac{1}{4})^2}{\pi(\frac{1}{4})^2} = (8c + 1)^2.$$

For edges $(\mathbf{u}_i, \mathbf{u}_{i+1})$ and $(\mathbf{u}_j, \mathbf{u}_{j+1})$ on P longer than $c/2^k$, one similarly obtains $|\mathbf{u}_i - \mathbf{u}_j| \geq 2^{-k-1}$ so that, here, Euclidean disks of radius 2^{-k-2} can be placed mutually disjoint within the total area of $\pi(2c + 2^{-k-2})^2$.

Theorem 4. *Let $G = (V, E)$ be a weak c -spanner with $V \subseteq \mathbb{R}^2$. Then G is a (C, δ) -power spanner for $\delta > 2$ where $C := (8c + 1)^2 \cdot \frac{(2c)^\delta}{1 - 2^{2-\delta}}$.*

Proof. Fix $\mathbf{u}, \mathbf{v} \in V$, w.l.o.g. $|\mathbf{u} - \mathbf{v}| = 1$. In the following we analyze the δ -cost of the path P constructed in Lemma 2 for $\delta = 2 + \epsilon$. We consider all edges on this path and divide them into classes depending on their lengths. By Lemma 3, there are at most $(8c + 1)^2$ edges of length between c and $2c$, each one inducing δ -cost at most $(2c)^\delta$. More generally, we have at most $(8c + 1)^2 \cdot 4^k$ edges of length between $c/2^k$ and $2c/2^k$ and the δ -cost of any such edge is at most $(2c/2^k)^\delta$. Summing up over all possible edges of P thus yields a total δ -cost of P of at most

$$\|P\|^\delta \leq \sum_{k=0}^{\infty} (8c + 1)^2 \cdot 4^k \cdot \left(\frac{2c}{2^k}\right)^\delta = (8c + 1)^2 \cdot \frac{(2c)^\delta}{1 - 2^{2-\delta}}$$

5.2 Weak Spanner Implies Power Spanner for $\delta = 2$

The preceding section showed that, for fixed $\delta > 2$, any weak c -spanner is also a (C, δ) -power spanner. The present section yields the same for $\delta = 2$, a case which, however, turns out to be much more involved. Moreover, our bounds on C in terms of c become slightly worse. In fact, the deepest result of this work is the following:

Theorem 5. *Let $G = (V, E)$ be a weak c -spanner with $V \subseteq \mathbb{R}^2$. Then G is a $(C, 2)$ -power spanner for $C := \mathcal{O}(c^8)$.*

Proof. First recall that between vertices $\mathbf{u}, \mathbf{v} \in V$ there is a path P in G from \mathbf{u} to \mathbf{v} which remains inside a square of length $\ell := 2c \cdot |\mathbf{u} - \mathbf{v}|$ and center \mathbf{u} . We denote such a square by $S_{\mathbf{u}}(\ell)$. By s we denote the starting point of the path and by t the end (target) point. We denote by $V(P)$ the vertex set of a path and by $E(P)$ the edge set of a path.

We give a constructive proof of the Lemma, i.e. given a path in G obeying the weak spanning property we construct a path which obeys the $(\mathcal{O}(c^8), 2)$ -power spanner property. For this we iteratively apply a procedure called **clean-up** to a path, yielding paths with smaller and smaller costs. Besides the path P in G this procedure has parameters $L, d, D \in \mathbb{R}^+$. Hereby, L denotes the edge length of a square with middle point s containing the whole path. The parameters d, D are in the range $0 < 3(c\sqrt{3}+2)d \leq D \leq L$

Procedure clean-up (P, L, d, D)

```

begin
  while three edges exist in  $P$  longer than  $2\sqrt{2}cd$ 
    starting or ending in the same cell of  $G_d$ 
  do
    Let  $C$  be such a cell in  $G_d$ 
     $P \leftarrow \mathbf{contract}(P, C)$ 
  od
  while there exists a cell in  $G_D$  where at least one vertex of  $P$ 
    lies in each of its  $G_d$ -sub-cells
  do
    Let  $C$  be such a cell of  $G_D$ 
    Let  $\text{rank}_P(u)$  denote the position of a vertex  $u$  in  $P$ 
    Sort all cells  $Z_1, \dots, Z_{(D/d)^2}$  of  $G_d$  in  $C$ 
      according to  $\min_{u \in Z_i \cap V(P)} \{\text{rank}_P(u)\}$ 
    Sort all cells  $Z'_1, \dots, Z'_{(D/d)^2}$  of  $G_d$  in  $C$ 
      according to  $\max_{u \in Z'_i \cap V(P)} \{\text{rank}_P(u)\}$ 
     $i \leftarrow 1$ 
    while cell  $Z_i$  is not horizontally neighbored
      to one of the cells  $\{Z'_1, \dots, Z'_i\}$ 
    nor  $Z'_i$  is horizontally neighbored
      to one of the cells  $\{Z_1, \dots, Z_i\}$ 
    do
       $i \leftarrow i + 1$ 
    od
    Let  $z$  and  $z'$  be the two neighbored cells
      from  $\{Z_1, \dots, Z_i\}$  and  $\{Z'_1, \dots, Z'_i\}$ 
     $P \leftarrow \mathbf{contract}(P, z \cup z')$ 
  od
  return  $P$ 
end

```

Procedure contract ($P = (v_1, \dots, v_m)$: path, A : area)

```

begin
  Let  $v_i$  be the first vertex of  $P$  in  $A$ 
  Let  $v_j$  be the last vertex of  $P$  in  $A$ 
  Let  $P' = (w_1, \dots, w_k)$  be a path between  $v_i = w_1$ 
    and  $v_j = w_k$  satisfying the weak spanner property
  return  $(v_1, \dots, v_{i-1}, w_1, \dots, w_k, v_{j+1}, \dots, v_m)$ 
end

```

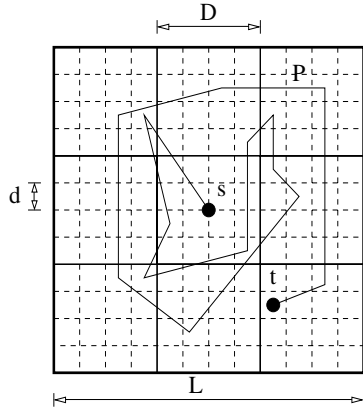


Fig. 4. The **clean-up** and the **contract** procedures and the idea for the Proof of Theorem 5

and can be chosen arbitrarily, yet fulfilling $D/d \in \mathbb{N}$ and $L/D \in \mathbb{N}$. These parameters define two edge-parallel grids G_d and G_D of grid size d and D such that boundaries of G_D are also edges of G_d . These grids fill out the square $S_u(L)$, while the boundary edge of $S_u(L)$ coincides with the boundary of G_d and G_D , see Fig. 4 The outcome of the procedure **clean-up** is a path $P' = \mathbf{clean-up}(P, L, d, D)$ which reduces the cost of the path while obeying other constraints, as we show shortly.

In Figure 4 we describe the procedure **clean-up** which uses the procedure **contract** described in Figure 4. Let $D(A)$ denote the diameter of the area A .

Lemma 4. *The procedure $P' = \mathbf{contract}(P, A)$ satisfies the following properties.*

- *Locality:* $\forall \mathbf{u} \in V(E(P) \setminus E(P')): \min_{\mathbf{p} \in A} |\mathbf{u} - \mathbf{p}| \leq c \cdot D(A)$ and $\max_{\mathbf{p} \in A} |\mathbf{u} - \mathbf{p}| \leq (c + 1) \cdot D(A)$.
- *Continuity of long edges:* $\forall e \in P' : |e| \geq 2c \cdot D(A) \implies e \in P$.

Proof. The maximum distance between v_i and v_j is at most $D(A)$. The replacement path (w_1, \dots, w_k) is inside a disk of radius $c \cdot D(A)$. Hence for all vertices \mathbf{u} of this replacement path we have $|\mathbf{u} - v_i| \leq cD(A)$ and therefore $\min_{\mathbf{p} \in A} |\mathbf{u} - \mathbf{p}| \leq |\mathbf{u} - v_i| \leq cD(A)$. From the triangle inequality it follows

$$\max_{\mathbf{p} \in A} |\mathbf{u} - \mathbf{p}| \leq D(A) + \min_{\mathbf{p} \in A} |\mathbf{u} - \mathbf{p}| \leq D(A) + |\mathbf{u} - v_i| \leq (c + 1)D(A).$$

The second property follows from the fact that all new edges inserted in P' lie inside a disk of radius $D(A)$.

Lemma 5. *For $D \geq 3(c\sqrt{3} + 2)d$ the procedure **clean-up** satisfies the four properties power efficiency, locality, empty space, and continuity of long edges.*

1. **Locality** For all vertices $\mathbf{u} \in V(P')$ there exists $\mathbf{v} \in V(P)$ such that

$$|\mathbf{u} - \mathbf{v}| \leq ((\sqrt{2} + \sqrt{3})c + 2) \cdot d.$$

2. **Continuity of Long Edges** For all edges e of P' with $|e| \geq 2\sqrt{3}cd$ it holds $e \in E(P)$.
3. **Power Efficiency** For all $k > 2\sqrt{3}c$:

$$\sum_{e \in E(P'): 2\sqrt{3}cd < |e| \leq kd} (|e|)^2 \leq k^2 d^2 \#F(P, G_d),$$

where $\#F(P, G_d)$ denotes the number of grid cells of G_d where at least one vertex of P lies which is the end point of an edge of minimum length $2\sqrt{3}d$.

4. **Empty Space** For all grid cells C of G_D we have at least one sub-cell of G_D within C without a vertex of P' .

Proof. All cells of G_d are called sub-cells in this proof for distinguishing them from the cells of G_D .

Observe that the **clean-up** procedure uses only contract-operation to change the path. As parameters for this procedure we use either a grid sub-cell C of edge length d and diameter $D(C) = \sqrt{2}d$ or two horizontally neighbored grid sub-cells Z and Z' with edge lengths d with diameter $D(Z \cup Z')$.

Further note, that in the first loop each sub-cell C of the grid G_d will be treated by the contract-procedure once. The reason is that from the contract procedures edges with lengths less than $2\sqrt{2}d$ are produced, while each sub-cell will loose all but two edges of P with minimum length $2\sqrt{2}d$. This also proves that the first loop always halts.

Now consider the second while-loop and concentrate on the part inside the loop before the contract-operation takes place. Since in every sub-cell of C we have a vertex of P we can compute the ordering Z_i and Z'_i as described by the algorithm. The main observation is that until the first two neighbored sub-cells Z and Z' from these sets are found, no two sub-cells Z and Z' from $Z \in \{Z_j\}_{j \leq i}$ and $Z' \in \{Z'_j\}_{j \leq i}$ are horizontally neighbored. Hence, in the $2d$ -surrounding of every point there is an empty sub-cell without any points of P .

The situation changes slightly if we apply the contract-operation. Then, an intermediate path will be added and possibly some of the empty sub-cells will start to contain vertices of the path. However, only sub-cells in a euclidean distance of $c\sqrt{3}d$ from sub-cells Z and Z' are affected by this operation. Now consider a square Q of $(c\sqrt{3} + 2)d \times (c\sqrt{3} + 2)d$ sub-cells in the middle of C . Then, at least two horizontally neighbored sub-cells will not be influenced by this contract-operation and thus remain empty.

One cannot completely neglect the influence of this operation to a neighbored grid cell of C . However, since $D \geq 3(c\sqrt{3} + 2)d$ the inner square Q is not affected by

contract-operation in neighbored grid cells of C because of the locality of the contract-operation.

This means if a cell C was object to the second while-loop, then an empty sub-cell will be produced which remains empty for the rest of the procedure. Hence, the second loop also terminates.

We now check the four required properties.

Locality. After the first loop the locality is satisfied even within a distance of $\sqrt{2}(c+1)d$. For this, observe that all treated cells contain end points of edges longer than $2\sqrt{2}cd$ which cannot be produced by contract-operations in this loop. Hence, if a cell is object to the contract-operation it was occupied by a vertex of P from the beginning. Then, from Lemma 4 it follows that for all new vertices of the path P there exists at least one old vertex in distance $\sqrt{2}(c+1)d$ after the first loop.

For the second loop we need to distinguish two cases. First, consider a cell C where in the inner square an empty sub-cell exists. In this case this cell will never be treated by this second loop. If new vertices are added to the path within this cell, then this will be caused by a contract-operation in a neighbored cell and will be considered in the second case.

Now consider all cells with preoccupied inner squares (preoccupation refers to the outcome of the first loop). These cells can be object to contract-operations of the second loop. However, they will add only vertices to their own sub-cells or to the outer sub-cells of neighbored cells. So, new vertices are added within a distance of $(\sqrt{3}c+1)d$ of vertices in the path at the beginning of the second loop. As we have seen above every such vertex is only $\sqrt{2}(c+1)d$ remote from an original vertex of a path. This gives a locality of distance $((\sqrt{3}+\sqrt{2})c+2)d$.

Continuity of Long Edges. Since the parametrized areas for the contract operation have a maximum diameter of $\sqrt{3}d$ this property follows directly from Lemma 4.

Power Efficiency. After the first loop the number of edges longer than $2\sqrt{2}cd$ is bounded by $\#F(P, G_d)$, because in every occupied sub-cell at most two edges start or end and each edge has two end points. Clearly, this number is an upper bound for edges longer than $2\sqrt{3}cd$. In the second loop no edges longer than $2\sqrt{3}cd$ will be added. This, directly implies the wanted bound.

Empty Space. As we have already pointed out the second loop always halts. Therefore the empty space property holds.

Lemma 6. *Given a path P_0 with source \mathbf{s} and target \mathbf{t} such that $\forall \mathbf{u} \in V(P_0) : |\mathbf{u} - \mathbf{s}| \leq c \cdot L$, where $L = c \cdot |\mathbf{u} - \mathbf{v}|$. Now iteratively apply $P_{i+1} = \text{clean-up}(P_i, L + \sum_{j=0}^i D_j, d_i, D_i)$ for $i = 1, 2, \dots$ where $D_i = L\beta^{1-i}$, $d_i = L\beta^{-i}$ for $\beta = 20\sqrt{3}c$. Then, P_m for $m = \lceil \log_\beta \min_{\mathbf{u}, \mathbf{v} \in V} |\mathbf{u} - \mathbf{v}| \rceil$ is path connecting \mathbf{s} to \mathbf{t} obeying the $(\mathcal{O}(c^8), 2)$ power spanner property.*

Proof. For this proof we make use of the four properties of the clean-up procedure. By assumption $c \geq \sqrt{3}$ we have $\beta \geq 60$.

First note that the the square of edge length L_i containing all vertices of path P_i can increase. However we can bound this effect by the locality property, giving $L_{i+1} \leq L_i + 2d_i$, where $d_i = L \cdot \beta^{-i}$. By assumption we have $c \geq \sqrt{3}$ and therefore $d_i \leq L4^{-i}$, which gives an upper bound of $L_i \leq 2L$ for all i .

Let $F_i = \#F(P_i, G_{d_i})$. Then $A_i = (d_i)^2 F_i$ denotes the area of all grid cells in G_{d_i} with a vertex of the path P_i which is the end point of an edge with length of at least $2\sqrt{3}d$. In the next iteration in each of this cells an empty space will be generated with an area of $(d_{i+1})^2$. Because of the locality property at least the following term of the edge length is subtracted

$$\sum_{j=1}^{\infty} 2\sqrt{3}d_{i+j} \leq \frac{1}{2}d_i .$$

Hence, an empty area of at least $\frac{1}{4}(d_i)^2$ remains after applying all clean-up-procedures. Let E_i be the sum of all these areas in this iteration. Therefore we have $A_i \leq 4\beta^2 E_i$. Clearly, these empty areas in this iteration do not intersect with empty areas in other areas (since they arise in areas which were not emptied before). Therefore all these spaces are inside the all-covering square of side length $2L$ yielding $\sum_{i=1}^{\infty} E_i \leq 4L^2$.

Because of the long edge continuity property, edges of minimum length $2\sqrt{3}d_i$ do not appear in rounds later than i . Therefore, the following sum S gives an upper bound on the power of the constructed path.

$$S = \sum_{i=1}^{\infty} \sum_{\substack{e \in E(P_i): \\ 2\sqrt{3}cd_i \leq |e| < 2\sqrt{3}c\beta d_i}} (|e|_2)^2 .$$

From the power efficiency property it now follows

$$\begin{aligned} S &\leq \sum_{i=1}^{\infty} 12c^2\beta^2 (d_i)^2 \#F(P_i, G_{d_i}) = 12c^2\beta^2 \sum_{i=1}^{\infty} (d_i)^2 F_i = 12c^2\beta^2 \sum_{i=1}^{\infty} A_i \\ &\leq 48c^2\beta^4 \sum_{i=1}^{\infty} E_i \leq 192 c^2\beta^4 L^2 \leq 192 c^4\beta^4 (|\mathbf{s} - \mathbf{t}|)^2 = O(c^8 (|\mathbf{s} - \mathbf{t}|)^2) \end{aligned}$$

This lemma completes the proof of the theorem.

5.3 Weak Spanner Does Not Imply Power Spanner for $\delta < 2$

Theorem 6. *To any $\delta < 2$, there exists a family of geometric graphs $G = (V, E)$ with $V \subseteq \mathbb{R}^2$ which are weak c -spanners for a constant c but no (C, δ) -power spanners for any fixed C .*

Proof. As $\delta < 2$, there is a $k \in \mathbb{R}$ such that $2 < k < 4^{1/\delta}$. We present a recursive construction (see Figure 5). Fix $\mathbf{u}^1 = (1/2, 1/2) \in \mathbb{R}^2$. In each following recursion step j , we replace every existing vertex $\mathbf{u}^i = (u_x^i, u_y^i)$ by four new vertices $\mathbf{u}^{4i-3} = (u_x^i - d, u_y^i + d)$, $\mathbf{u}^{4i-2} = (u_x^i + d, u_y^i + d)$, $\mathbf{u}^{4i-1} = (u_x^i + d, u_y^i - d)$, and $\mathbf{u}^{4i} = (u_x^i - d, u_y^i - d)$ where $d := 1/(2k^j)$. Finally, we consider the graph $G_j := (V_j, E_j)$

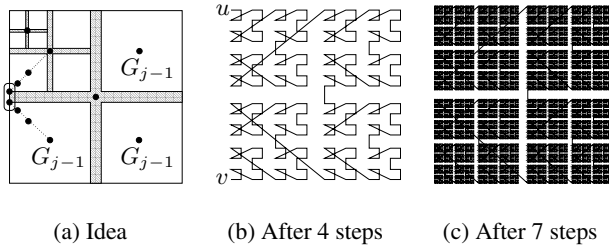


Fig. 5. Recursive construction: The underlying idea and two examples for $k = 2.1$

with $V_j := \{\mathbf{u}^i \mid i \in \{1, \dots, 4^j\}\}$ and $E_j := \{(\mathbf{u}^i, \mathbf{u}^{i+1}) \mid i \in \{1, \dots, 4^j - 1\}\}$. The resulting graph after 4 recursion steps with $k = 2.1$ is given in Figure 5(b). Let $\mathbf{u} = \mathbf{u}^1$ and $\mathbf{v} = \mathbf{u}^{4^j}$.

Lemma 7. *The graph G_j is a weak c -spanner for $c := \frac{\sqrt{2k(k-1)}}{k-2}$ independent of j .*

Proof. We prove the claim by induction over j . For $j = 1$ the weak stretch factor is dominated by the path between \mathbf{u} and \mathbf{v} . The distance between \mathbf{u} and \mathbf{v} is $1/k$. The farthest vertex on the path from \mathbf{u} to \mathbf{v} is \mathbf{u}_3 . It holds that $|\mathbf{u} - \mathbf{u}_3| \leq \sqrt{2}/k$. Hence, we get the weak stretch factor $\sqrt{2} \leq \frac{\sqrt{2k(k-1)}}{k-2} = c$. Now, we consider G_j for any j . We can divide the graph G_j into four parts G_j^1, \dots, G_j^4 . By the definition of our recursive construction each part equals the graph G_{j-1} . For two vertices in one part the required weak c -spanner property holds by induction. We have to concentrate on two vertices which are chosen from two different parts. Since G_j^i is connected to G_j^{i+1} it is sufficient to consider a vertex from G_j^1 and a vertex from G_j^4 . On the one hand, the weak stretch factor is affected by the shortest distance between such chosen vertices. On the other hand, this distance is given by (see also Figure 5(a))

$$\left(\frac{1}{2} \cdot \left(1 + \frac{1}{k} - \sum_{i=2}^j \left(\frac{1}{k}\right)^i\right) - \frac{1}{2}\right) \cdot 2 \geq \frac{k-2}{k(k-1)}$$

The entire construction lies in a bounded square of side length 1, and hence we get a weak stretch factor of at most $\frac{\sqrt{2k(k-1)}}{k-2} = c$.

Lemma 8. *The graphs G_j are no (C, δ) -power spanners for any fixed C .*

Proof. It suffices to consider the δ -cost of the path from \mathbf{u} to \mathbf{v} . The direct link from \mathbf{u} to \mathbf{v} has δ -cost at most 1. For any path P from \mathbf{u} to \mathbf{v} in G , it holds that

$$\|P\|^\delta \geq 3 \cdot 4^j \cdot \left(\left(\frac{1}{k}\right)^j\right)^\delta = 3 \cdot \left(\frac{4}{k^\delta}\right)^j$$

which goes to infinity if $j \rightarrow \infty$ for $k < 4^{1/\delta}$.

Combining Lemma 7 and Lemma 8 proves Theorem 6.

5.4 Fractal Dimension

The present section generalizes the construction and analysis used in Lemma 8. To this end, consider a self-similar polygonal fractal curve Γ as the result of repeated application of some generator K being a polygonal chain with starting point \mathbf{u} and end point \mathbf{v} . This is illustrated in Fig. 2 showing a generator (left) and the resulting fractal curve (right part); see also [14, 2]. But other examples are plenty: the KOCH Snowflake or the space filling HILBERT Curve (Fig. 6). Recall that the fractal dimension of Γ is defined as

$$\frac{\log(\text{number of self-similar pieces})}{\log(\text{magnification factor})}$$

Theorem 7. *Let K be a polygonal chain, Γ_n the result of n -fold application of K , and Γ the final self-similar polygonal fractal curve with dimension d . Then, for all $\delta < d$, there is no fixed C such that Γ_n is a (C, δ) -power spanner for all n .*

Proof. Let p denote the number of self-similar pieces in Γ_n and m the magnification factor. Then by definition, we have $d = \log(p) / \log(m)$. Now consider the δ -cost of the (unique) path P in Γ_n from \mathbf{u} to \mathbf{v} . Since Γ_n is constructed recursively we get in the n -th step:

$$\|P\|^\delta = p^n \cdot \left(\frac{1}{m}\right)^{\delta n} = \left(\frac{p}{m^\delta}\right)^n$$

Note that $\|P\|^\delta$ is unbounded iff $p/m^\delta > 1$, that is, iff $\delta < \log(p) / \log(m) = d$.

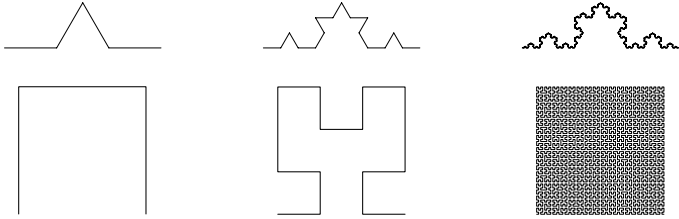


Fig. 6. Two Generators and the Fractal Curves they induce due to KOCH and HILBERT

The fractal dimensions of the KOCH and HILBERT Curves are well-known. Therefore by virtue of Theorem 7, the KOCH Curve is not a (\cdot, δ) -power spanner for any $\delta < \log(4) / \log(3) \approx 1.26$; similarly, HILBERT's Curve is not a (\cdot, δ) -power spanner for any $\delta < 2$. One can show that KOCH's Curve is a weak spanner (the proof is analogous to Theorem 1). However HILBERT's Curve is no weak spanner as its inner vertices come arbitrarily close to each other. Further examples for self-similar polygonal curves, e.g., SIERPINSKIS's triangle, can be found in [14, 2].

6 Power Spanner Hierarchy

In the following we show that for $\Delta > \delta > 0$, a (c, δ) -power spanner is also a (C, Δ) -power spanner with C depending only on c and Δ/δ . Then we show that the converse fails in general by presenting to each $\Delta > \delta > 0$ a family of graphs which are (c, Δ) -power spanners for some constant c but no (C, δ) -power spanners for any fixed C .

Theorem 8. *Let $G = (V, E)$ be a (c, δ) -power spanner with $V \subseteq \mathbb{R}^2$, $0 < \delta < \Delta$. Then G is also a (C, Δ) -power spanner for $C := c^{\Delta/\delta}$.*

Proof. Let $\mathbf{u}, \mathbf{v} \in V$ be two arbitrary vertices. Since G is a (c, δ) -power spanner there exists a path $P = (\mathbf{u} = \mathbf{u}_1, \dots, \mathbf{u}_l = \mathbf{v})$ with $\|P\|^\delta = \sum_{i=1}^{l-1} |\mathbf{u}_i - \mathbf{u}_{i+1}|^\delta \leq c \cdot |\mathbf{u} - \mathbf{v}|_2^\delta$. The function $f(x) = x^{\Delta/\delta}$ being convex on $[0, \infty[$, one may apply JENSEN'S Inequality:

$$\|P\|^\Delta = \sum_{i=1}^{l-1} |\mathbf{u}_i - \mathbf{u}_{i+1}|^\Delta = \sum_{i=1}^{l-1} \left(|\mathbf{u}_i - \mathbf{u}_{i+1}|^\delta \right)^{\Delta/\delta} \leq \left(\sum_{i=1}^{l-1} |\mathbf{u}_i - \mathbf{u}_{i+1}|^\delta \right)^{\Delta/\delta} \leq c^{\Delta/\delta} \cdot |\mathbf{u} - \mathbf{v}|^\Delta.$$

Theorem 9. *Let $0 < \delta < \Delta$. There is a family of geometric graphs which are (c, Δ) -power spanners but no (C, δ) -power spanners for any fixed C .*

Proof. We slightly modify the construction from the proof of Theorem 3 by placing n vertices $\mathbf{u} = \mathbf{u}_1, \dots, \mathbf{u}_n = \mathbf{v}$ on an appropriately scaled circle such that the Euclidean distance between \mathbf{u} and \mathbf{v} is 1 and $|\mathbf{v}_i - \mathbf{v}_{i+1}| = (1/i)^{1/\delta}$ for all $i = 1, \dots, n - 1$. Now, in the graph $G = (V, E)$ with edges $(\mathbf{v}_i, \mathbf{v}_{i+1})$, the unique path P from \mathbf{u} to \mathbf{v} has Δ -cost

$$\|P\|^\Delta = \sum_{i=1}^{n-1} (1/i)^{\Delta/\delta} \leq \sum_{i=1}^{\infty} (1/i)^{\Delta/\delta} =: c$$

a convergent series since $\Delta/\delta > 1$. This is to be compared to the cost of the direct link from \mathbf{u} to \mathbf{v} which amounts to 1 both w.r.t. Δ and δ . On the other hand, the δ -cost of P is given by the harmonic series $\sum_{i=1}^{n-1} (1/i)^{\delta/\delta} = \Theta(\log n)$ and thus cannot be bounded by any C independent of n .

7 Higher-Dimensional Case

For simplicity, most results in this work have been formulated for the case of (not necessarily planar) geometric graphs in the plane. They immediately apply to higher dimensions as well, however with the exception of Section 5 (Weak versus Power Spanners). In fact, similar techniques yield that, for instance in 3D, each weak c -spanner is a (C, δ) -power spanner for $\delta \geq 3$ with C depending only on c and δ whereas to any $\delta < 3$, there are counter-examples of weak c -spanners that are not (C, δ) -power spanners for any fixed C ; analogously in higher dimensions.

8 Conclusions

We investigate the relations between spanners, weak spanners, and power spanners. In the plane, for $\delta \geq 2$ it turns out that being a spanner is the strongest property, followed by being a weak spanner and finally being a (\cdot, δ) -power spanner. For $1 < \delta < 2$, spanner is still strongest whereas weak spanner and (\cdot, δ) -power spanner are not related to each other. For $0 < \delta < 1$ finally, (\cdot, δ) -power spanner implies both spanner and weak spanner. For higher dimensions, similar relations/independencies hold. All stretch factors in these relations are constant and are pairwise polynomially bounded.

In [9, 7] a geometric graph called **YY** or **SparsY-Graph** was investigated as a topology for wireless networks. It is constructed by dividing the area around each vertex into $k \in \mathbb{N}$ non-overlapping sectors or cones of angle $\theta = 2\pi/k$ each. In each sector of a vertex, there is at most one outgoing edge and if there is one, then this goes to the nearest neighbor in this sector. A vertex accepts in each of its sectors only one ingoing edge and this must be the shortest one in this sector. For this graph the relation between weak and power spanner is exemplarily investigated in [9, 15] by performing experiments on uniformly and randomly distributed vertex sets. They conclude that the **SparsY-Graph** might be a spanner and also a power spanner. The first conjecture is still open, while the latter was independently proven in [8] and [7].

Observe that **SparsY** is well-known to yield a good weak spanner already for $k > 6$ [7]. Regarding that our Theorem 5 asserts *any* weak spanner to be also a (\cdot, δ) -power spanner for $\delta \geq 2$, this includes the above result *and* weakens the presumption from $k \geq 120$ [8] to $k > 6$.

Although our results are exhaustive with respect to the different kinds of geometric graphs and in terms of δ , one might wonder about the optimality of the bounds obtained for C 's dependence on c ; for instance: Any c -spanner is a (C, δ) -power spanner for $C = c^\delta$, $\delta > 1$; and this bound *is* optimal. But is there some $C = o(c^4)$ such that any weak c -spanner is a (C, δ) -power spanner as long as $\delta > 2$? Is there some $C = o(c^8)$ such that any weak c -spanner is a $(C, 2)$ -power spanner?

References

1. K. Alzoubi, X.-Y. Li, Y. Wang, P.J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.
2. D. Eppstein. The Geometry Junkyard: Fractals, www.ics.uci.edu/~eppstein/junkyard/fractal.html.
3. D. Eppstein. Beta-skeletons have unbounded dilation. Technical Report ICS-TR-96-15, 1996.
4. D. Eppstein. Spanning trees and spanners. In *Handbook of Computational Geometry*, pages 425–461. 2000.
5. M. Fischer, T. Lukovszki, and M. Ziegler. Geometric searching in walkthrough animations with weak spanners in real time. In *6th Annual European Symposium on Algorithms (ESA '98)*, pages 163–174, 1998.
6. M. Fischer, F. Meyer auf der Heide, and W.-B. Strothmann. Dynamic data structures for realtime management of large geometric scenes. In *5th Annual European Symposium on Algorithms (ESA '97)*, pages 157–170, 1997.

7. M. Grünewald, T. Lukovszki, C. Schindelhauer, and K. Volbert. Distributed Maintenance of Resource Efficient Wireless Network Topologies (Ext. Abstract). In *8th EURO-PAR'02*, pages 935–946, 2002.
8. L. Jia, R. Rajaraman, and C. Scheideler. On local algorithms for topology control and routing in ad hoc networks. In *Proc. 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA '03)*, pages 220–229, 2003.
9. X.-Y. Li, P.-J. Wan, and Y. Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN01)*, pages 564–567, 2001.
10. F. Meyer auf der Heide, C. Schindelhauer, K. Volbert, and M. Grünewald. Energy, congestion and dilation in radio networks. In *Proc. 14th Symposium on Parallel Algorithms and Architectures (SPAA'02)*, pages 230–237, 2002.
11. F. Meyer auf der Heide, C. Schindelhauer, K. Volbert, and M. Grünewald. Congestion, Dilation, and Energy in Radio Networks. *Theory of Computing Systems*, 37(3):343–370, 2004.
12. R. Rajaraman. Topology control and routing in ad hoc networks: a survey. *SIGACT News*, 33(2):60–73, 2002.
13. S. B. Rao and W. D. Smith. Approximating geometrical graphs via spanners and banyans. In *Proceedings of the 30th annual ACM symposium on Theory of computing*, pages 540–550, 1998.
14. C. Tricot. *Curves and Fractal Dimension*. Springer, 1995.
15. K. Volbert. Experimental Analysis of Adjustable Sectorized Topologies for Static Ad Hoc Networks, accepted for DIALM-POMC 2004.
16. Y. Wang and X.-Y. Li. Distributed Spanner with Bounded Degree for Wireless Ad Hoc Networks. In *Parallel and Distributed Computing Issues in Wireless networks and Mobile Computing*, page 120, 2002.
17. Y. Wang, X.-Y. Li, P.-J. Wan, and O. Frieder. Sparse power efficient topology for wireless networks. In *Proc. ACM Hawaii International Conference on System Sciences (HICSS'02)*, page 296, 2002.
18. A. C.-C. Yao. On Constructing Minimum Spanning Trees in k -dimensional space and related problems. *SIAM J. Comput.*, 11:721–736, 1982.

Techniques for Indexing and Querying Temporal Observations for a Collection of Objects*

(Extended Abstract)

Qingmin Shi and Joseph JaJa

Institute of Advanced Computer Studies, University of Maryland,
College Park, MD 20742, USA
{joseph, qshi}@umiacs.umd.edu

Abstract. We consider the problem of dynamically indexing temporal observations about a collection of objects, each observation consisting of a key identifying the object, a list of attribute values and a timestamp indicating the time at which these values were recorded. We make no assumptions about the rates at which these observations are collected, nor do we assume that the various objects have about the same number of observations. We develop indexing structures that are almost linear in the total number of observations available at any given time instance, and that support dynamic additions of new observations in polylogarithmic time. Moreover, these structures allow the quick handling of queries to identify objects whose attribute values fall within a certain range at every time instance of a specified time interval. Provably good bounds are established.

1 Introduction

Consider the scenario in which temporal observations about a large collection of objects are being collected asynchronously. Each observation record consists of a key identifying the object, a list of the values of a number of attributes, and a timestamp indicating the time at which these particular values were recorded. We make no assumptions about the rates at which these observations are collected. We allow the collection of objects to vary with time, with possibly new objects added to our collection at any time instances. The only assumption we make is that the timestamp of a new observation record for a given object has to be larger than the timestamp of the object's observations that are already stored in our data structure.

This type of data appears in many real world applications. For example, in the Great Duck Island (GDI) system [13] for monitoring the microclimates in and around nesting burrows used by the Leach's Storm Petrel, various environmental

* Supported in part by the National Science Foundation through the National Partnership for Advanced Computational Infrastructure (NPACI), DoD-MD Procurement under contract MDA90402C0428, and NASA under the ESIP Program NCC5300.

conditions such as temperature, photoresistor, and barometric pressure are measured by a collection of sensors at different geolocations and sent to a PostgreSQL database for storage and analysis. The Automated Local Evaluation in Real-Time (ALERT) system [1] periodically collect meteorological/hydrological data such as water level, temperature, and wind speed at ALERT sensor sites, which is then transmitted to the base station for processing and web-based querying. More and more car-rental companies have installed GPS systems on its fleet of vehicles, which report back the speed and location of these vehicles.

While queries on “snapshots” of the temporal data have been studied for some time (see for example [22] for a survey), the problem of quickly detecting the temporal patterns of such data collections remains open. We believe that a crucial first step towards solving this problem is to develop techniques to support the discovery of a set of core temporal patterns. In this paper, we define the *conjunctive temporal range search problem*, which is an extension of the well-known and fundamental range search problem and develop efficient techniques to handle such a problem.

A conjunctive temporal range search problem involves dynamically maintaining an indexing structure to enable quick identification of objects whose attributes fall within a set of value ranges during a given time interval. Note that queries of this type are seemingly similar but actually very different from the following *disjunctive temporal range query*: identify objects whose attributes fall within a set of value ranges at a time instance during a given time interval. Since we will not address the disjunctive temporal range queries in this paper (results on this type of queries will be mentioned in Section 2), we will use the term *conjunctive temporal range query* to refer to conjunctive temporal range query for simplicity.

Applications involving temporal data can benefit from an efficient solution to this problem. For example, in a flood warning system, identifying locations where the water level are above certain threshold during a limited time period could be very important; and a car rental company many wish to identify the vehicles in its fleet that have been driven within the boundary of a certain geographical area during a specified time interval.

In this paper, we develop efficient indexing structures to handle the conjunctive temporal range queries which achieve provably good performance bounds in terms of space, querying time, and update time.

1.1 Problem Definition and Main Results

Consider a set S of n objects $\{O_1, O_2, \dots, O_n\}$, each identified by a key o_i and characterized by a set of d attributes $\{v_{i,1}(t), v_{i,2}(t), \dots, v_{i,d}(t)\}$ whose values change over time t . Observations about each object are collected at discrete time instances. Let m_i be the number of observations about object O_i , say collected at time instances $t_i^1 < t_i^2 < \dots < t_i^{m_i}$. We denote the observations of O_i at t_i^j as a vector $\mathbf{v}_i^j = [v_{i,1}^j, v_{i,2}^j, \dots, v_{i,d}^j]$, where $v_{i,l}^j = v_{i,l}(t_i^j)$ for $l = 1, \dots, d$. The total number of observations for all the objects in S is $m = \sum_{i=1, \dots, n} m_i$. We

denote the number of distinct time instances among $\{t_i^j | 1 \leq i \leq n, 1 \leq j \leq m_i\}$ as $m' \leq m$.

A temporal range query is formally defined as follows:

Given two vectors $\mathbf{a} = [a_1, a_2, \dots, a_d]$ and $\mathbf{b} = [b_1, b_2, \dots, b_d]$, and a time interval $[t_s, t_e]$, determine the set Q of objects such that $O_i \in Q$ if and only if the following two conditions are satisfied:

- $\exists j$ such that $t_i^j \in \{t_i^l | l = 1, \dots, m_i\}$ and $t_s \leq t_i^j \leq t_e$, i.e., there is at least one observation of O_i recorded between t_s and t_e .¹
- $\forall j$ such that $t_i^j \in \{t_i^l | l = 1, \dots, m_i\}$ and $t_s \leq t_i^j \leq t_e$, we have $a_k \leq v_{i,k}^j \leq b_k$ for all $1 \leq k \leq d$.

We will call each such object a *relevant* object with respect to the query.

Let f denote the output size of such a query. The main results of this paper are summarized as the following four theorems. The first three theorems deal with the case of a single variable, while Theorem 4 deals with the general case.

Theorem 1. n
 m $O(\log m \log^2 n + f)$
 $O(m \log n \cdot \min\{\log m, n\})$
 $O(m \log m \log^2 n)$ $O(\log m \log^2 n)$

Theorem 2. n
 m $O(\log m (\log^2 n + f))$
 $O(m \log n)$ $O(m \log m \log^2 n)$
 $O(\log m \log^2 n)$

Theorem 3. n
 m $O(\log^4 m + f)$
 $O(m \log m)$
 $O(m \log^2 m)$ $O(\log^3 m)$

Theorem 4. n m
 m'
 $O(\log^{2d-1} n + f)$ $O(m \log^{2d-2} n + m')$
 $O(m \log^{2d-1} n + m')$
 $O(\log^{2d-1} n + \Delta m')$ $\Delta m'$

2 Previous Related Work

A related class of problems that have been studied in the literature, especially the database literature (see for example [12, 15, 18, 26, 27]), deals with temporal

¹ This condition can be removed if necessary by introducing a simple additional data structure [25] based on priority search trees [16].

data by appending a time interval to each piece of data separately, thus treating each observation, rather than each object, as an individual entity. These techniques work well for queries that involve only a single time instance but are ill suited for the temporal range queries studied here because applying them directly would result in algorithms whose execution time depends on the number of time instances in the query time interval, which is undesirable for our general problem.

The disjunctive temporal range queries can be reduced to the so called *range query* (see [10, 9, 8, 3, 17, 24] for results on this topic), and seems to be easier than the conjunctive temporal range queries. Indeed, we have shown in [24] that a one-sided disjunctive temporal range query on single-attribute objects can be handled optimally in $O(\log m + f)$ time using linear space. Similar to disjunctive temporal range queries is the problem of indexing moving objects (see for example [2, 11, 21, 5, 20]). We should emphasize again that a conjunctive temporal range query requires every observation of a proper object in the time interval to fall within the value ranges while a disjunctive temporal range query needs only one such observation.

Finally, a “synchronized” and static version of this problem, in which observations of the objects are all collected at the same sequence of time instances and are available beforehand, was studied in our previous paper [23].

3 One-Sided Temporal Range Queries: The Static Case

3.1 Preliminaries

Let v_i^j denote the observation of object O_i at time instance t_i^j . Given a query represented by the triple (t_s, t_e, a) , we aim at identifying the objects that have at least one observation during the time interval $[t_s, t_e]$ and whose observations within that time interval are all greater than or equal to a . We call this type of queries *one-sided temporal range queries*. We will give three solutions to this problem, each providing a tradeoff between the storage cost and the query time.

We start by making the following straightforward observation.

Observation 1. *Let O_i be an object with observations v_i^j at time instances t_i^j . Then O_i satisfies the query (t_s, t_e, a) if and only if $\min\{v_i^j \mid t_s \leq t_i^j \leq t_e\} \geq a$.*

Note that we define $\min\{v_i^j \mid t_s \leq t_i^j \leq t_e\} = -\infty$ whenever no j exists such that $t_s \leq t_i^j \leq t_e$. We define the *open time interval* $I_i^j = (s_i^j, e_i^j)$ of observation v_i^j as the longest open time interval during which v_i^j is the smallest observation of O_i . More specifically, Let $v_i^{j_1}$ be the latest observation such that $j_1 < j$ and $v_i^{j_1} \leq v_i^j$ and $v_i^{j_2}$ be the earliest observation such that $j_2 > j$ and $v_i^{j_2} < v_i^j$. Then $s_i^j = j_1$ and $e_i^j = j_2$. If j_1 does not exist, then $s_i^j = -\infty$. Similarly, $e_i^j = +\infty$ if j_2 does not exist. We thus transform an observation v_i^j into a 5-tuple (or tuple for short) $(v_i^j, t_i^j, s_i^j, e_i^j, o_i)$.

The following lemma is critical to our approach.

Lemma 1. *Let O_i be an object and (t_s, t_e, a) be a query. Let $(v_i^j, t_i^j, s_i^j, e_i^j, o_i)$ be a tuple such that $t_i^j \in [t_s, t_e]$ and $v_i^j \geq a$.*

By definition, an object O_i is proper only if no observation is smaller than a during the time interval $[t_s, t_e]$. Let $v_i^j = \min\{v_i^l | t_s \leq t_i^l \leq t_e\}$ (it always exists for a proper object), where j is the smallest such index if multiple minima exist. It is obvious that the tuple $(v_i^j, t_i^j, s_i^j, e_i^j, o_i)$ satisfies the three conditions stated in the lemma. On the other hand, if O_i is not proper, then either there is no observation of O_i in $[t_s, t_e]$, or the value of at least one such observation is less than a . In the latter case, no interval (s_i^l, e_i^l) with $t_i^l \in [t_s, t_e]$ and $v_i^l \geq a$ will be able to cover $[t_s, t_e]$. The uniqueness of this tuple is due to the fact that the dominant intervals are maximal. \square

3.2 An $O(m \log m)$ -Space $O(\log n \log m + f)$ -Query Time Solution

We call the data structure proposed in this section the *Fast Time Reporting* because it is the fastest among the three solutions proposed; and the one discussed in Section 3.3, which uses less space but requires more query time, is called the *Fast Space Reporting*.

Let $(t_1, t_2, \dots, t_{m'})$ be the sorted list of all the distinct time instances. The skeleton of the FTR-tree is a balanced binary tree T built on this list. Each node u is associated with a set $S(u)$ of up to n tuples (n is the number of objects). If u is the k th leaf starting from the left, then $S(u) = \{(v_i^j, t_i^j, s_i^j, e_i^j, o_i) | t_i^j = t_k\}$. If u is an internal node with two children v and w , we decide for each object O_i which tuple to be added to $S(u)$ by examining the tuples corresponding to O_i in v and w . If $S(v)$ and $S(w)$ do not contain any such tuple, then no tuple for O_i will be added to $S(u)$. If only one of them do, then that tuple is included in $S(u)$. If both of them do, then the tuple with the longest dominant interval is chosen. Note that in this case the longer interval always contains the shorter one. Figure 1 illustrates how the tuples associated with each node are collected for an example consisting of two objects and a total of 16 observations.

Given a query (t_s, t_e, a) , we can easily find the set of at most $2(\log m' - 1)$ nodes in T that correspond to the interval $[t_s, t_e]$. An allocation node is a node whose corresponding time interval is fully contained in $[t_s, t_e]$ and that of whose parent is not. For each allocation node v , we only need to report those tuples in $S(v)$ that satisfy $(s_i^j, e_i^j) \supset [t_s, t_e]$, and $v_i^j \geq a$. Lemma 1 guarantees that exactly one such tuple will be reported for each proper object. No further search on v 's descendants is necessary.

For each allocation node v , looking for tuples $(v_i^j, t_i^j, s_i^j, e_i^j, o_i)$ that satisfy $(s_i^j, e_i^j) \supset [t_s, t_e]$ and $v_i^j \geq a$ is equivalent to a three dimensional dominance reporting problem, which can be solved in $O(\log n(v) + f(v))$ time and $O(n(v))$ space using the data structure of Makris and Tsakalidis [14], which we will refer to as the *Fast Reporting*, where $n(v)$ is the number of tuples stored in v and $f(v)$ is the number of tuples reported.

We show in the full paper [25] that this scheme achieves the following.

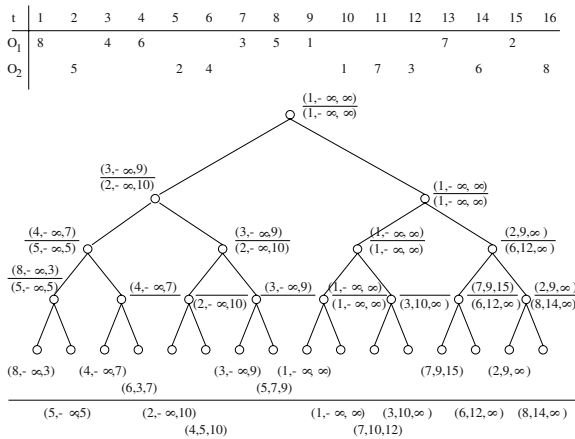


Fig. 1. The observations of two objects and the corresponding FTR-tree. Each node is associated with up to two tuples, the one above the horizontal line corresponds to object O_1 and the one below it corresponds to object O_2 . We omit the values of t_i^j and o_i for each tuple

Theorem 5. $O(m \min(\log m, n))$ space complexity for a query.

3.3 An $O(m)$ -Space $O(\log m(\log n + f))$ -Query Time Solution

In a FTR-tree, a tuple could be stored at multiple levels of the primary tree T . In this section, we propose the CTR-trees by removing these duplicates. Consider an arbitrary observation v_i^j stored in an node u of T . We stipulate that v_i^j be removed from u if there is no observation of O_i stored in the sibling of u . It is easy to verify that the modified data structure uses $O(m)$ space.

To ensure the correctness of our answer to a query, we modify the algorithm of Section 3.2 to search not only the allocation nodes, but also the nodes on the path from the root to them. Although no proper object will be missed in this process, some tuples that do not satisfy the conditions stated in Lemma 1 may be mistakenly reported. Fortunately, these tuples always correspond to proper objects. The only side effect is that an object may be reported up to $O(\log m)$ times, once at each level of T .

It is easy to see that the total number of nodes on the paths from the root to the allocation nodes is $O(\log m)$, and $O(\log n)$ time is taken at each of them. Thus we have the following theorem.

Theorem 6. $O(m)$ space complexity and $O(\log m \log n + f \log m)$ query time complexity.

3.4 An $O(m)$ -Space $O(\log^3 m + f)$ -Query Time Solution

In this section, we give a linear space solution that reports each proper object exactly once. We call it *linear space solution* (). In designing the LTR-tree, we apply twice the interval tree techniques of Edelsbrunner [7] and use dominance trees to handle the queries.

Rewriting the conditions stated in Lemma 1, we have $t_s \in (s_i^j, t_i^j]$, $t_e \in [t_i^j, e_i^j)$, and $v_i^j \geq a$. Each tuple $(v_i^j, t_i^j, s_i^j, e_i^j, o_i)$ can be viewed as a rectangular plate in a three-dimensional space whose edges are parallel to the x- and y-axes, and whose projections to these two axes are $(s_i^j, t_i^j]$ and $[t_i^j, e_i^j)$ respectively; and the query can be viewed as finding the plates that are intersected by a ray perpendicular to the x-y plane shooting in the direction of positive z-axis from the point (t_s, t_e, a) .

We consider the projections of these plates to the x-y plane, which are rectangles. The primary structure of the LTR-tree is an interval tree T on the set of intervals $\{(s_i^j, t_i^j] | i = 1, \dots, n; j = 1, \dots, m_i\}$, thus distributing the rectangles to the nodes of T . Let $S(v)$ be the set of rectangles associated with a node v of T . We build a secondary interval tree $T(v)$ on $S(v)$, this time based on vertical edges of the rectangles in $S(v)$. By doing so, we further distribute the tuples in $S(v)$ to the nodes of $T(v)$. For the tuples associated with each node μ of $T(v)$, we construct four versions $T_0(\mu)$, $T_1(\mu)$, $T_2(\mu)$, and $T_3(\mu)$ of the dominance tree, for the following point dominance queries respectively: 1. $(t_s > s_i^j, t_e \geq t_i^j, v_i^j \geq a)$; 2. $(t_s > s_i^j, t_e < e_i^j, v_i^j \geq a)$; 3. $(t_s \leq t_i^j, t_e \geq t_i^j, v_i^j \geq a)$; 4. $(t_s \leq t_i^j, t_e < e_i^j, v_i^j \geq a)$.

To answer a query (t_s, t_e, a) , we start from the root r of the primary tree. We first access $T(r)$ to report tuples stored at r and then check if $t_s \leq x(r)$. If this is the case, we recursively access the subtree rooted at r 's left child; otherwise, we recursively access the subtree rooted at r 's right child. When accessing a secondary tree $T(u)$, we start from its root α . We first compare t_e with $y(\alpha)$. Depending on whether $t_s \leq x(u)$ and whether $t_e < y(\alpha)$, we access one of the four dominance trees associated with α . More specifically, we access $T_0(\alpha)$ if $t_s \leq x(u)$ and $t_e < y(\alpha)$; $T_1(\alpha)$ if $t_s \leq x(u)$ and $t_e \geq y(\alpha)$; $T_2(\alpha)$ if $t_s \geq x(u)$ and $t_e < y(\alpha)$; and $T_3(\alpha)$ if $t_s \geq x(u)$ and $t_e \geq y(\alpha)$. After the points associated with α are reported, we recursively access its left child if $t_e \leq y(\alpha)$ or its right child otherwise.

In the full paper [25], we provide the correctness proof of this scheme and show that it has the complexity bounds stated in the following theorem.

Theorem 7. *There is a linear space solution that reports all proper objects in n objects, m intervals, and f queries in $O(\log^3 m + f)$ time.*

4 One-Sided Temporal Range Queries: The Dynamic Case

In this section, we consider the problem of designing dynamic indexing structures that enable the quick handling of temporal range queries and at the same time can be efficiently updated when new observations are added. As stated before,

we make the assumption that the timestamp of a new observation of an object O_i is larger than that of any existing observations of O_i . Note that adding a new object simply means adding the first observation of that object.

4.1 Creating and Updating Tuples

The addition of new observations may require that many of the existing tuples corresponding to the same object be updated to reflect the possible change of their dominant intervals. To facilitate the quick identification of such tuples, we maintain a Cartesian tree [28] C_i for each object O_i . A Cartesian tree for a sequence $(t_i^j, v_i^j), 1 \leq j \leq m_i$, is a binary tree with m_i nodes. The root stores the smallest value v_i^j over the time interval $[t_i^1, t_i^{m_i}]$, where j is the smallest such index if multiple minima exist. Its left child is the root of the Cartesian tree for observations $\{v_i^1, \dots, v_i^{j-1}\}$; and its right child is the root of the Cartesian tree for observations $\{v_i^{j+1}, \dots, v_i^{m_i}\}$. Note that a node may not have a left or right child. The Cartesian tree C_i can be built in $O(m_i \log m_i)$ time by inserting the observations in order of their timestamps, using the algorithm we discuss next.

Let $v_i^{m_i+1}$ be the new observation of object O_i with a timestamp $t_i^{m_i+1}$, where $t_i^{m_i+1} > t_i^{m_i}$. Let Π_i be the rightmost path of the Cartesian tree C_i before the addition of $v_i^{m_i+1}$ and π_i be the prefix of Π_i such that each node on π_i , except the root, is the right child of its parent. To update C_i , we first find the pair of parent-child nodes u_{pred} and u_{succ} on π_i and the corresponding observations $v_i^{j_{\text{pred}}}$ and $v_i^{j_{\text{succ}}}$ such that $v_i^{j_{\text{pred}}} \leq v_i^{m_i+1} < v_i^{j_{\text{succ}}}$. u_{pred} (u_{succ}) is null if $v_i^{m_i+1}$ is less than (greater than or equal to) all the observations associated with π_i . Since the values of the observations corresponding to the nodes on π_i are non-decreasing from the root, we can easily find this pair in $O(\log m_i)$ time using binary search.

Now consider nodes u_{pred} and u_{succ} . If u_{pred} is not null, then the new node u that corresponds to $v_i^{m_i+1}$ becomes its right child. If u_{succ} is not null, it becomes the left child of u . The node u becomes the root of the new C_i if u_{pred} is null.

The following lemma guarantees that on average, only a constant number of tuples need to be updated.

Lemma 2. *Let m_i be the number of observations of object O_i and g be the number of observations of other objects. Then the number of tuples that need to be updated when a new observation $v_i^{m_i+1}$ is added to O_i is $O(\log m_i + g)$.*

(.) Consider the insertion of a new observation $v_i^{m_i+1}$ of O_i . We only need to update the tuples whose corresponding observations are associated with the suffix of π_i , starting from u_{succ} . Let h_i^j be the length of π_i in the version of C_i with j observations and let l_i^j be the length of the path from u_{pred} to the root (the height of the root is 1). Then $h_i^j - l_i^j$ existing tuples need to be updated. These tuples can be retrieved in $O(1)$ time each, provided that u_{succ} has been located, a task that can be done in $O(\log h_i^j)$ time. Furthermore, the length h_i^{j+1} of the new π_i

after the insertion is $l_i^j + 1$. Adding up $h_i^j - l_i^j$ for $j = m_i, \dots, 2m_i - 1$ gives the lemma. \square

The following lemma gives the aggregate number of tuples that need to be updated over a sequence of adding new observations.

Lemma 3. *Let m be the number of tuples in a dynamic dominance tree T and let k be the number of new tuples inserted into T . Then the total number of tuples that need to be updated is $O(m + k)$.*

Lemma 3 allows us to handle the addition by first identifying the old tuples that need to be updated, followed by performing each of the updates, and finally inserting the new tuple.

4.2 Dynamic Data Structures for 3-D Dominance Queries

We present in this section a data structure called *dynamic dominance tree* that supports 3-D dominance report queries and dynamic insertion and deletion. We now elaborate on this data structure using the version of dominance query in which we are asked to find all the points $p = (p_x, p_y, p_z)$ that are dominated by a query point $q = (q_x, q_y, q_z)$, i.e., $p_x \leq q_x, p_y \leq q_y$, and $p_z \leq q_z$.

Given a set of n 3-D points, we first build a *dynamic dominance tree* [4] T of degree c on the z -coordinates sorted in increasing order, where c is a constant. For each internal node v , we build a priority search tree [16] that stores the set of points in the subtree of v projected onto the x - y plane. A dominance query can be answered by first identifying the $O(c \log n)$ allocation nodes in T that together correspond to the z -range $(-\infty, q_z]$, and then searching the corresponding priority search trees to answer the query $(p_x \leq q_x; p_y \leq q_y)$.

Insertion and deletion on dynamic dominance tree are very similar to the ones described in [4] and thus is omitted here. Generalization of the above results to higher dimensions is straightforward and is summarized by the following lemma.

Lemma 4. *Let $d \geq 2$ be the dimensionality of the data. Then the dynamic dominance tree T can be built in $O(n \log^{d-2} n)$ time and space, and it can answer a dominance query in $O(n \log^{d-1} n)$ time. The dynamic dominance tree T can be built in $O(n \log^{d-1} n)$ time and space, and it can answer a dominance query in $O(\log^{d-1} n + f)$ time, where f is the number of points in the subtree of T that are dominated by the query point.*

4.3 Dynamic FTR-Tree

To make the structure in Section 3.2 dynamic, we replace the binary tree built on the time instances by a weight-balanced B-tree T of degree c . Each node is associated with a set of tuples, each representing an object. The dominant interval of a tuple associated with an internal node v covers the dominant intervals of all the tuples stored in the subtree of v representing the same object. With each node v of T , we store the dynamic dominance tree structure $T_{\text{dom}}(v)$ built on the tuples stored at v , and a dynamic binary search tree, say a red-black tree [6], $T_{\text{key}}(v)$ built on the keys associated with these tuples. It can be shown

using similar arguments as in the static case that the size of this data structure is $O(m \log n \min\{\log m, n\})$ (the extra $\log n$ factor is due to the dynamic dominance tree structure being used).

The query process is almost the same as in the static case. The only difference is that we now have up to $O(c \log m)$ allocation nodes. Each such node v takes $O(\log^2 n + f(v))$ time to search.

There are two major steps required to update our overall data structure. The first is to update the tuples that are no longer valid, and the second is to insert the new time stamp and the new tuple into the primary tree.

Consider the update step. Suppose that the tuple $(v_i^l, t_i^l, s_i^l, e_i^l, o_i)$ needs to be updated. Notice that the entry t_i^l of this tuple does not change. Therefore, there is no need to update the primary tree. Furthermore, we have the following lemma.

Lemma 5.

Let v be a node in the primary tree. Then $T_{\text{dom}}(v) \subseteq T_{\text{key}}(v)$.

Therefore, what we need to do is to go through each node on the path from the root to the leaf node corresponding to t_i^l . For each node v on this path, we search $T_{\text{key}}(v)$ using o_i to find the old tuple and replace it with the new one. Then we remove the same old tuple from, and insert the new tuple into, $T_{\text{dom}}(v)$. The whole process takes $O(\log m \log^2 n)$ time.

To insert a new tuple $(v_i^{j+1}, t_i^{j+1}, s_i^{j+1}, +\infty, o_i)$, we first insert the new time instance into the primary tree T . This may cause up to $O(\log m)$ nodes to split, which can be handled in $O(\log m \log^2 n)$ amortized time following similar arguments as in Section 3.2. To insert the new tuple, we traverse the path from the leaf node corresponding to t_i^{j+1} up toward the root. At each node v visited, we search the representative tuple of O_i in $T_{\text{key}}(v)$ using o_i . If there is no such tuple, we insert the new tuple into both $T_{\text{key}}(v)$ and $T_{\text{dom}}(v)$. If one such tuple is found, we check if it needs to be replaced by the new tuple. If it does, then we remove the old tuple from and insert the new tuple into both $T_{\text{key}}(v)$ and $T_{\text{dom}}(v)$. Otherwise, we do not need to visit any of v 's ancestors. The full proof of Theorem 1 is given in [25].

In the full paper [25], we show how to use similar techniques to show Theorems 2 and 3 about the CTR- and the LTR-trees.

5 Handling the General Temporal Queries

For the general problem, we assume that we have a predefined time hierarchy imposed on our time line, say starting at a fixed time instance t_0 until $t_{m+1} = +\infty$, such that all queries involve one of the time intervals defined in this hierarchy. This is indeed the case in many applications such as OLAPs. We are interested in queries that will identify objects whose attributes fall within certain ranges at every time instance in one of the time intervals defined by the hierarchy.

We formally define the time hierarchy as a tree $T = (V, E)$. Each node v of T is associated with a time interval $I(v) = [t_s, t_e)$ at a certain level of this hierarchy. An internal node v has a set of children that correspond to the time intervals of a finer granularity. Except for the root, which is associated with the time interval $[t_0, +\infty)$, the time interval associated with any other internal node v is $I(v) = \bigcup_{u \in \text{children}(v)} I(u)$. The leaves correspond to time intervals of the finest granularity in the hierarchy. A temporal range query on this time hierarchy is defined as follows.

Given two vectors $\mathbf{a} = [a_1, a_2, \dots, a_d]$ and $\mathbf{b} = [b_1, b_2, \dots, b_d]$, and a node $v \in V$, determine the set Q of objects such that $O_i \in Q$ if and only if the following two conditions are true:

- There exist at least one observation taken at time $t_i^j \in I(v)$.
- For every observation \mathbf{v}_i^j such that $t_i^j \in I(v)$, we have $a_k \leq v_{i,k}^j \leq b_k$ for $k = 1, 2, \dots, d$.

We store at each node v a set $S(v)$ of $(2d+1)$ -tuples: $S(v) = \{(\min_{i,1}^v, \max_{i,1}^v, \dots, \min_{i,d}^v, \max_{i,d}^v, o_i) | \exists j, t_i^j \in I(v)\}$, where $\min_{i,l}^v$ and $\max_{i,l}^v$ are the minimum and maximum values of the l th attribute of O_i during the time interval $I(v)$. If there is no observation for O_i during $I(v)$, then there is no tuple in $S(v)$ representing O_i . To be able to tell which objects are represented in v , we maintain a tree $T_{\text{key}}(v)$ to index the tuples in $S(v)$ on the keys o_i .

By Observation 1, we can answer a query by determining the $(2d+1)$ -tuples at v which satisfy: $\max_{i,1}^v \leq b_l$ and $\min_{i,1}^v \geq a_l$, for all $l = 1, 2, \dots, d$. Finding such tuples in $S(v)$ is equivalent to answering a $(2d)$ -dimensional dominance query, which can be handled in $O(\log^{2d-1} n + f(v))$ time using a data structure $T_{\text{dom}}(v)$ of size $O(n \log^{2d-2} n)$. The total number of tuples stored in T is $O(m)$, since each tuple is stored in a constant number of nodes, one at each level of the hierarchy (the number of hierarchy levels is assumed to be a constant independent of the number of observations). Let $n(v)$ be the number of tuples stored in v . The overall size of the data structure is $O\left(\sum_{v \in V} n(v) \log^{2d-2} n(v) + m'\right) = O\left(m \log^{2d-2} n + m'\right)$, where m' is the number of leaves in T , which is typically much smaller than m . The construction of this data structure is straightforward. We first use $O(m)$ time to construct the set $S(v)$ for each node v . We then spend $O(n(v) \log n(v))$ time to build $T_{\text{key}}(v)$ and $O(n(v) \log^{2d-1} n(v))$ time to build $T_{\text{dom}}(v)$. The overall preprocessing time is $O(m \log^{2d-1} n + m')$.

Details of adding new observations and the full proof of Theorem 4 are provided in the full paper [25].

References

- [1] <http://www.alertsystems.org/>.
- [2] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *19th ACM Symposium on Principles of Database Systems*, pages 175–186, 2000.
- [3] P. K. Agarwal, S. Govindarajan, and S. Muthukrishnan. Range search in categorical data: color range searching on grid. In *ESA'02*, pages 17–28, 2002.

- [4] L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory. In *FOCS'96*, pages 560–569, Oct. 1996.
- [5] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with SETI. In *1st Biennial Conf. on Innovative Data Systems Research*, 2003.
- [6] Cormen, Leiserson, and Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [7] H. Edelsbrunner. A new approach to rectangle intersections, part I. *Int. J. Computer Mathematics*, 13:209–219, 1983.
- [8] P. Gupta, R. Janardan, and M. Smid. Computational geometry: generalized intersection searching. In *Handbook of Data Structures and Applications*. CRC Press.
- [9] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19:282–317, 1995.
- [10] R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3(1):39–69, 1993.
- [11] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *PODS'99*, pages 261–272, 1999.
- [12] S. Lanka and E. Mays. Fully persistent B⁺-trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 426–435, 1991.
- [13] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, Sept. 2002.
- [14] C. Makris and A. K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.
- [15] Y. Manolopoulos and G. Kapetanakis. Overlapping B⁺-trees for temporal data. In *Proc. of the 5th Jerusalem Conf. on Info. Tech.*, pages 491–498, 1990.
- [16] E. M. McCreight. Priority search trees. *SIAM J. Computing*, 14(2):257–276, 1985.
- [17] C. W. Mortensen. Generalized static orthogonal range searching in less space. Technical Report TR-2003-22, The IT University of Copenhagen, 2003.
- [18] M. A. Nascimento and J. R. O. Silva. Towards historical R-trees. In *Proceedings of the ACM Symposium on Applied Computing*, pages 235–240, Feb. 1998.
- [19] M. H. Overmars. *The design of dynamic data structures*. Springer-Verlag, LNCS 156, 1983.
- [20] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB'00*, pages 395–406, Sept. 2000.
- [21] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 331–342, 2000.
- [22] B. Salzberg and V. J. Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys*, 31(2):158–221, 1999.
- [23] Q. Shi and J. JaJa. Fast algorithms for a class of temporal range queries. In *WADS'03*, pages 91–102, Ottawa, Canada, 2003.
- [24] Q. Shi and J. JaJa. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. Technical Report CS-TR-4542, Institute of Advanced Computer Studies (UMIACS), University of Maryland, 2003.
- [25] Q. Shi and J. JaJa. Techniques for indexing and querying temporal observations for a collection of objects. Technical Report CS-TR-4503, Institute of Advanced Computer Studies (UMIACS), University of Maryland, 2003.
- [26] Y. Tao and D. Papadias. Efficient historical R-trees. In *Proceedings of the 13th Int. Conf. on Scientific and Statistical Database Management*, pages 223–232, 2001.

- [27] T. Tzouramanis, Y. Manolopoulos, and M. Vassilakopoulos. Overlapping Linear Quadrees: A spatio-temporal access method. In *Proc. of the 6th ACM Symp. on Advances in Geo. Info. Systems (ACM-GIS)*, pages 1–7, Bethesda, MD, 1998.
- [28] J. Vuillemin. A unifying look at data structures. *CACM*, 23(4):229–239, 1980.

Approximation Algorithms for the Consecutive Ones Submatrix Problem on Sparse Matrices*

Jinsong Tan and Louxin Zhang

Department of Mathematics, National University of Singapore,
2 Science Drive 2, Singapore 117543
{mattjs, matzlx}@nus.edu.sg

Abstract. A 0-1 matrix has the Consecutive Ones Property (C1P) if there is a permutation of its columns that leaves the 1's consecutive in each row. The Consecutive Ones Submatrix (COS) problem is, given a 0-1 matrix A , to find the largest number of columns of A that form a submatrix with the C1P property. Such a problem has potential applications in physical mapping with hybridization data. This paper proves that the COS problem remains NP-hard for i) $(2, 3)$ -matrices with at most two 1's in each column and at most three 1's in each row and for ii) $(3, 2)$ -matrices with at most three 1's in each column and at most two 1's in each row. This solves an open problem posed in a recent paper of Hajiaghayi and Ganjali [12]. We further prove that the COS problem is 0.8-approximatable for $(2, 3)$ -matrices and 0.5-approximatable for the matrices in which each column contains at most two 1's and for $(3, 2)$ -matrices.

1 Introduction

A 0-1 matrix is said to have the Consecutive Ones Property (C1P) if there is a permutation of its columns that leaves the 1's consecutive in each row. This property was first mentioned by an archaeologist named Petrie in 1899 [15]. In the last three decades, the study of the C1P property for a given 0-1 matrix found different applications in graph theory [7, 15], computer science [5, 9], and genome sequencing [20].

For sequencing large genomes, biologists first construct a clone library consisting of thousands of clones. Each clone represents a DNA fragment of small size. Then, they map each clone and assemble these maps to determine the map of the entire genome using overlapping between the clones (overlapping can be achieved by using a few restriction enzymes). One approach to determine whether two clones overlap or not is based on hybridization with short probes. In this approach, a clone is exposed to a number of probes and which of these probes hybridize to the clone is determined. Therefore, for the map assembly problem

* The research was partially supported by the Biomedical Research Council of Singapore (BMRC01/1/21/19/140).

with m clones and n probes, the experimental data is a $n \times m$ matrix $A = (a_{ij})$, where $a_{ij} = 1$ if clone c_j hybridizes with probe p_i , and $a_{ij} = 0$ otherwise. If the probes are STS's, each probe is unlikely to occur twice in the genome and hence this leads to physical mapping with unique probes [6]. If an experimental data matrix A is error-free, it has the C1P property and there are efficient algorithms for constructing a correct ordering of clones and probes [7, 3, 2]. However, in reality, physical mapping is prone to various errors and the physical mapping problem with error and uncertainties becomes extremely difficult [2, 14, 10, 19]. This motivates us to study the following problem

Consecutive Ones Submatrix (COS)

Instance: A 0-1 matrix A .

Question: Find the largest number of columns of A that form a submatrix with C1P property.

The decision version of this problem is one of the earliest NP-complete problems appearing in the Garey and Johnson's book [8]. However, the NP-completeness proof was misreferred to in the book. Recently, Hajiaghayi and Ganjali gave a proof [12]. Actually, they proved that the COS problem is NP-complete even for 0-1 matrices in which there are at most two 1's in each column and at most four 1's in each row. On the other hand, the COS problem can be solved in polynomial time for 0-1 matrices with at most two 1's in each row and column. Therefore, their work raises the problem of weather the COS problem remains NP-complete or not for i) (3, 2)-matrices which have at most two 1's in each row and at most three 1's in each column and for ii) (2, 3)-matrices which have at most three 1's in each row and at most two 1's in each column.

Studying the COS problem for matrices with a small number of 1's in column and/or rows is not only theoretically interesting, but also practically important. Actually, this sparsity restriction was proposed by Lander and Istrail (personal communication) with hopes of making the mapping problem tractable. This paper is divided into five sections. We answer the open problem by proving its NP-completeness for the two special cases just mentioned (in Section 2). Furthermore, we prove that the COS problem is 0.8-approximatable for (2, 3)-matrices, 0.5-approximatable for the matrices in which each column contains at most two 1's (in Section 3) and for (3, 2)-matrices (in Section 4). Finally, we conclude this work with several remarks in Section 5. Two closely related works are that the different versions of physical mapping with errors are showed to be NP-hard for sparse matrices in [1, 21].

For basic concepts and knowledge in NP-hardness, polynomial time approximation, we refer readers to the book [8] by Garey and Johnson.

2 NP-Hardness of the Problems

In this section, we study the decision version of the COS problem: given a 0-1 matrix A and a positive integer K , are there K columns of A that form a submatrix with the C1P property. A 0-1 matrix is a (\dots) if it has at

most two 1's in each column and at most three 1's in each row; similarly, it is a $(2, 3)$ -matrix if it has at most three 1's in each column and at most two 1's in each row.

2.1 COS for $(2, 3)$ -Matrices Is NP-Complete

To prove the COS problem NP-complete for $(2, 3)$ -matrices, we first define a special spanning tree problem and prove it NP-complete. Formally, it is defined as follows.

Definition 1. A graph $G = (V, E)$ is called a $(2, 3)$ -graph if $\deg(v) \leq 3$ for every $v \in V$ and $|E| \geq c - 2$, where $c \geq 3$.

Spanning Caterpillar Tree in Degree-3 Graph

Instance: A graph $G = (V, E)$ in which each node has degree at most 3.

Question: Does G contain a spanning caterpillar subtree T ?

Lemma 1. A graph $G = (V, E)$ is a $(2, 3)$ -graph if and only if it contains a spanning caterpillar subtree.

The proof is by a reduction from the Hamiltonian Path problem for cubic graphs in which every node has degree 3 (see the comment in GT39 in [8]).

Given a cubic graph $G = (V, E)$, we construct a new graph $G' = (V', E')$ by inserting a new node in each edge of G . Formally, we have

$$V' = V \cup \{x_{uv} \mid (u, v) \in E\}$$

$$E' = \{(u, x_{uv}), (x_{uv}, v) \mid (u, v) \in E\},$$

where x_{uv} is called an *edge node*. The reduction follows from the fact that there exists a Hamiltonian path in G if and only if there exists a spanning caterpillar subtree of G' . We conclude the proof by proving this fact as follows.

The ‘only if’ direction is easily seen. If there is a Hamiltonian path P in the cubic G , we obtain a desired spanning caterpillar subtree of G' by replacing each edge (u, v) by a two-edge path $ux_{uv}v$ if (u, v) is in the path P and attaching the inserted node w_{uv} to u otherwise.

For the ‘if’ direction, we assume that such a spanning caterpillar subtree T exists in G' . By construction of G' , any degree-3 node in T must be a degree-3 node in G' and hence corresponds to a node in G . This also implies that each leaf adjacent to a degree-3 node in T must be an edge node. Therefore, by removing the leaves adjacent to degree-3 nodes from T , we obtain a path P' . Obviously, by construction, P' corresponds to a Hamiltonian path in G . □

Next, we proceed to show the NP-completeness of the COS problem for $(2, 3)$ -matrices by a reduction from the Spanning Caterpillar Tree in Degree-3 Graph problem.

Theorem 1. The COS problem for $(2, 3)$ -matrices is NP-complete.

The proof is just a refinement of the reduction used to prove the *NP*-completeness of the COS problem in [12]. Given a $G = (V, E)$ with n nodes and e edges in which each node has degree at most 3, we consider its incidence matrix $B(G) = (b_{ij})$. Recall that $B(G)$ has a row for each node $v_i \in G$ and a column for each edge $e_j \in G$; and b_{ij} is 1 if v_i is incident to e_j in G and 0 otherwise. Since each node in G has degree at most 3, the incidence matrix $B(G)$ has exact two 1's in each column and at most three 1's in each row.

It can be easily seen that a subset of columns corresponds to a subgraph induced by the corresponding edges in G . Moreover, we claim that G has a spanning caterpillar subtree if and only if $B(G)$ has a submatrix of size $n \times (n - 1)$ that has C1P property. We conclude the proof by proving this claim.

We start with the ‘only if’ direction. Suppose G has a spanning caterpillar subtree T in which each degree-3 node is adjacent to at least one leaf. Since G has n nodes, T contains $n - 1$ edges. Moreover, the submatrix induced by the columns corresponding to these $n - 1$ edges satisfies C1P property. To prove this fact, we consider two cases.

If T is just a Hamiltonian path $u_1, u_2, u_3, \dots, u_n$ of G , then, the columns corresponding to edges $(u_1, u_2), (u_2, u_3), \dots, (u_{n-1}, u_n)$ (in this order) form an $n \times (n - 1)$ submatrix in which all the 1's are consecutive in each row.

If T contains degree-3 nodes, then, we consider the a longest path in T . Assume such a longest path P is $u_1, u_2, u_3, \dots, u_m$, where $m < n$. Since G has n nodes and T is a spanning caterpillar subtree, P contains exactly $n - m$ nodes $u_{i_1}, u_{i_2}, \dots, u_{i_{n-m}}$ of degree 3 in T such that each u_{i_j} is adjacent to a unique leaf v_{i_j} that is not in the path P . By arranging columns corresponding to edges $(u_1, u_2), (u_2, u_3), \dots, (u_{m_1}, u_m)$ in the same order and inserting the column corresponding to (u_{i_j}, v_{i_j}) between those corresponding to (u_{i_j-1}, u_{i_j}) and (u_{i_j}, u_{i_j+1}) for each j , we obtain an $n \times (n - 1)$ submatrix in which all the 1's are consecutive in each row. This is because the row corresponding to a leaf contains only one 1.

Now we show the ‘if’ direction. Suppose there exists a submatrix of size $n \times (n - 1)$ that has C1P property in $B(G)$. The subgraph induced by the corresponding $n - 1$ edges must not admit a cycle (otherwise, at least one of the n rows cannot satisfy the consecutive ones property). Hence, the $(n - 1)$ edges induce a spanning subtree of G since these edges are incident to n nodes. Suppose T does not satisfy our special requirement that each degree-3 node in it is adjacent to a leaf node. Then, there exists a node $v \in T$ such that v is adjacent to three nodes $v_1, v_2, v_3 \in T$ with $degree(v_i) \geq 2$, $i = 1, 2, 3$. It's easy to see that there is no way to arrange the columns corresponding to $(v_1, v), (v_2, v)$ and (v_3, v) in $B(G)$ such that all the four rows corresponding to v, v_1, v_2 and v_3 satisfy C1P property. Therefore, the spanning subtree T is caterpillar, i.e, each degree-3 node in it is adjacent to at least a leaf.

□

2.2 COS for (3, 2)-Matrices Is NP-Complete

In this subsection, we prove the NP-completeness of the COS problem for (3, 2)-matrices by a reduction from the following NP-complete problem for cubic graphs.

Induced Disjoint-Path-Union Subgraph

Instance: Graph $G = (V, E)$ in which each node has degree 3, and a positive integer $k \leq |V|$.

Question: Does there exist a k -node subset $V' \subseteq V$ that induces a subgraph $G(V')$ (not necessarily connected) whose components are paths?

Lemma 2.

According to the theorems by Yannakakis [22] and Lewis [16] (see also problem GT21 and the related comment on page 195 in the book [8] by Garey and Johnson), the problem of finding an induced subgraph with property Π of a particular given size in a cubic graph is NP-complete if Π is:

- 1) *Nontrivial.* Π is true for a single node and not satisfied by all the graphs in the given domain;
- 2) *Interesting.* There are arbitrarily large graphs satisfying Π ;
- 3) *Easy.* For a given graph, the property can be verified in polynomial time;
- 4) *Hereditary.* If a graph satisfies property Π , then any of its induced subgraph must also satisfy property Π .

We now prove our problem is NP-complete by showing that our property π of being a disjoint union of paths satisfies the above conditions.

Let \mathcal{G} denote the set of all cubic graphs. The set \mathcal{G}' of graphs that are isomorphic to an induced subgraph of G is the domain of our property π .

It is easy to see that the property π holds for a single node and an arbitrarily large graph in the domain, but not all graphs in the domain; also, π is hereditary since whenever G is a disjoint union of paths, so is its any subgraph. In addition, the property π can be easily verified in polynomial time. Hence, our problem is NP-complete. □

Theorem 2.

We prove this NP-completeness result by giving a simple reduction from the Induced Disjoint-Path-Union Subgraph problem for cubic graphs.

Given a cubic graph G with n nodes and m edges, we consider the transpose $B'(G) = (b_{ij})$ of the incidence matrix of G . Thus, $B'(G)$ is a $m \times n$ matrix in which each row corresponds to an edge of G and each column a node of G . The entry b_{ij} is 1 if edge e_i has v_j as an end node; it is 0 otherwise. Since G is a cubic graph, $B'(G)$ has exactly three 1's in each column and two 1's in each row.

We claim that a k -node subset V' induces a subgraph $G(V')$ of G that is a disjoint union of paths if and only if the k columns corresponding to nodes in V' form an $m \times k$ submatrix with C1P property.

The ‘only if’ direction is similar to Case 1 in the ‘only if’ condition proof in Theorem 1, so we focus on the ‘if’ direction. Suppose $B'(G)$ contains an $m \times k$ submatrix C with C1P property. Let V' be the subset of nodes corresponding to the k columns in C . We show that V' induces a subgraph with the desired property. Let v be a node in V' . Since G is cubic, we assume the neighbors of v are v_1 and v_2 and v_3 in G . If all its neighbors v_1, v_2 and v_3 are also in V' , then, there is no way to arrange these four columns corresponding to v, v_1, v_2 and v_3 so that the two 1’s are consecutive in rows corresponding to (v_1, v) and (v_2, v) and (v_3, v) . (Note that each column in the submatrix C contains exactly three 1’s.) Hence, at most two of v_i ’s are in V' . This implies that V' induces a subgraph in which each node has degree at most 2. Furthermore, it is easy to see that the induced subgraph $G(V')$ does not contain a cycle (otherwise, at least one of the rows corresponding to the edges in the cycle cannot have its 1’s consecutive under any column permutation). This concludes the proof. \square

3 Approximation Algorithms for $(2, \Delta)$ -Matrices

We present a 0.8-approximation algorithm for the COS problem for $(2, 3)$ -matrices: Given a $(2, 3)$ -matrix A , find a largest submatrix B of A consisting of a subset of A ’s columns with the C1P property. We also show that a direct generalization of the algorithm turns out to have an approximation ratio 0.5 for $(2, \Delta)$ -matrices. Without loss of generality, we may assume a $(2, \Delta)$ -matrix $A = (a_{ij})$ satisfies the following properties in this section:

1. Row-distinguishability. No two rows are identical.
2. Column-distinguishability. No two columns are identical.
3. Connectedness. For any partition of the rows into non-empty subsets R' and R'' , there are $i' \in R'$ and $i'' \in R''$ such that $a_{i'j} = a_{i''j} = 1$ for some column j .

Recall that a $(2, \Delta)$ -matrix A has at most Δ 1’s in each row and at most two 1’s in each column. If A contains any column with only one 1, we expand A into a ‘full’ $(2, \Delta)$ -matrix A' whose columns contains exactly two 1’s as follows. For each column j containing only one 1, we add an extra row i' that has 1 at the column j and 0 elsewhere. Assume A has k columns with a single 1. Then, its expansion A' has the same number of columns as A and k more rows than A . Finally, we assume A has n columns. For any subset $C \subseteq \{1, 2, \dots, n\}$, we use $A(C)$ to denote the submatrix of A consisting of columns with indices in C . Then, we give the following simple observation without proof.

Proposition 1. For any subset $C \subseteq \{1, 2, \dots, n\}$, $A(C)$ has the C1P property if and only if $A'(C)$ has the C1P property.

By Proposition 1, the COS problem has the same solution to A and A' , and a good approximation solution to A' is also a good approximation to A . Therefore, assume that A has exactly two 1’s in each column. Obviously, such

a $(2, \Delta)$ -matrix A defines uniquely a graph $G(A) = (V, E)$ that has A as the incidence matrix: Each row and column of A corresponds to a node and edge in $G(A)$, respectively. We assume $E = \{1, 2, \dots, n\}$. For a subset $C \subseteq E$, the subgraph of $G(A)$ induced by C has node set V and edge set C and is denoted by $G_C(A)$.

Proposition 2. For $C \subseteq \{1, 2, \dots, n\}$, $A(C)$ has the C1P property if and only if $G_C(A) = (V, C)$ is a union of caterpillar subtrees.

(Sketch of Proof) For convenience, we use the term edges and nodes in $G(A)$ and columns and rows in A interchangeably. Suppose $G_C(A)$ is a union of caterpillar subtrees C_1, C_2, \dots, C_h . For each $1 \leq t \leq h$, using the same discussion as in the proof of Theorem 1, the edges in $C_t = (V_t, E_t)$ form a submatrix $A_{V_t \times E_t}(C)$ with the C1P property. By arranging the columns in each caterpillar subtree in a block and arranging the columns within each block according to their connection in the corresponding subtree, we obtain a submatrix in which all the 1's in each row are arranged consecutively. This is because $A(i, j) = 0$ if node i and edge j don't belong to the same caterpillar tree. Hence, $A(C)$ has the C1P property.

Conversely, suppose $A(C)$ has the C1P property. Using the same discussion as in the proof of Theorem 1, each component of $G_C(A)$ must be a caterpillar subtree. Therefore, $G_C(A)$ is a union of caterpillar trees. □

3.1 A 0.8-Approximation Algorithm for $(2, 3)$ -Matrices

Theorem 3. Let A be a $(2, 3)$ -matrix. Then, A has a 0.8-approximating solution if and only if A has the C1P property.

Let the $(2, 3)$ -matrix A have m rows and n columns. Then, $G(A) = (V, E)$ has m nodes and n edges. Since each row contains at most three 1's in A , each node has degree at most 3.

For any subset $C \subseteq \{1, 2, \dots, n\}$, by Proposition 2, if $A(C)$ has the C1P property, $G_C(A)$ is a union of caterpillar subtrees. Therefore, $G_C(A)$ has at most $m - 1$ edges, and hence C contains at most $(m - 1)$ columns.

To find a 0.8-approximating solution to the COS problem for A , we first find a spanning tree T in $G(A)$. Then, we apply ALGORITHM A given below to T to find a union of caterpillar subtrees that have at least $0.8(m - 1)$ edges. The set of edges in the union gives a desired solution by Proposition 2. To describe ALGORITHM A, we recall some basic concepts in graph theory. In a rooted tree, the *depth* of a node is equal to the distance between the root and itself. A non-leaf node is said to be *internal*. For an internal node x in a rooted tree T' , we use $T'(x)$ to denote the subtrees rooted at x and $p(x)$ to denote the *parent* of x that is the first node on the unique path from x to the root. Finally, an internal node is said to be *complete internal* if it is adjacent to 3 internal nodes. Obviously, a complete internal node is of degree 3.

ALGORITHM A

Input: A tree T with m nodes, in which each node has degree at most 3;
 Output: A union of caterpillar subtrees with at least $0.8(m - 1)$ edges.

- 1 Pick a leaf r of T and root T at r . Let T_r be this rooted tree;
- 2 Initially, set $RT = T_r$, $RE = \phi$;
- 3 Do a Breath First Search on T_r , for each node v encountered;
- 4 Record its depth in T_r ;
- 5 If v is *complete* in T_r , *push* it to the end of list L_c
- 6 Repeat the following action until RT contains at most 6 nodes:
- 7 *Pop* a node x from the end of L_c ;
- 8 if x is *complete* in RT
- 9 Remove the edge between x and its parent $p(x)$;
- 10 $RT = RT - RT(x) - (x, p(x))$, $RE = RE \cup \{(x, p(x))\}$;
- 11 Output $T - RE$;

It is easy to see that the algorithm takes $O(m)$ time. Notice that a tree with at most 6 nodes is caterpillar if each node has degree at most 3. Now we prove its connectedness. Now consider a complete internal node x with a largest depth in the tree RT (Note each node *popped* from list L_c at line 7 is with a largest depth in RT). First, the subtree rooted at x , $RT(x)$, is a caterpillar subtree since by assumption every internal node of it is adjacent to at most two internal nodes. Second, since x is a complete internal node, there are at least two internal nodes in $RT(x)$ and hence $RT(x)$ contains at least 4 edges. This implies that each repeat of line 6-10 removes at least 5 edges (including $(x, p(x))$). Therefore, line 6-10 of the algorithm can be repeated at most $(m - 1)/5$ times and at most $0.2(m - 1)$ edges are removed from the input tree. □

3.2 0.5-Approximation Algorithm for $(2, \Delta)$ -Matrices

Now we show that a direct generalization of ALGORITHM A has approximation ratio 0.5 for $(2, \Delta)$ -matrices.

Let A be a $(2, \Delta)$ -matrix with m rows and n -columns. Then, the corresponding graph $G(A)$ has m nodes and n edges, and each node of it has degree at most Δ . We propose the following generalization to ALGORITHM A.

First, we find a spanning tree T of $G(A)$ and root it at a leaf r . Let the resulting rooted tree be T_r . Obviously, each node in T_r has degree at most Δ . Recall that a non-leaf node is called an internal node. For an internal node x , we use l_x to denote the number of leaves adjacent to it and d_x to denote its degree; x is said to be *complete* if it is adjacent to at least three internal nodes, i.e., $d_x - l_x \geq 3$. We find a union of caterpillar subtrees of T (hence $G(A)$) by repeating the following action until no complete internal nodes exist in T_r :

Pick a complete internal node x in T_r with the largest depth and then remove the edge $(x, p(x))$ and any other $d_x - l_x - 3$ edges between x and its internal neighbors.

In each repeat, we take away at least $l_x + 1 + 2(d_x - l_x - 1) = 2d_x - l_x - 1$ edges from T_r by removing $d_x - l_x - 2$ edges. Thus, at least half of the edges in T remain in the output union of caterpillar subtrees. Hence we proved that

Theorem 4.

4 A 0.5-Approximation Algorithm for (3, 2)-Matrices

In this section, we use a partition theorem of Lovász in graph theory to obtain a 0.5-approximation algorithm for the COS problem for (3, 2)-matrices. The idea is also generalized to $(\Delta, 2)$ -matrices.

4.1 The Algorithm

Recall that a (3, 2)-matrix contains at least three 1's in each column and at most two 1's in each row. Noting that any column permutation preserves the consecutiveness of 1's in a row with at most one 1, we only focus on the (3, 2)-matrices which have exactly two 1's in each row in the rest of this section.

Let A be such a (3, 2)-matrix. Then, it defines uniquely a graph $G(A)$ with maximum degree 3 in which nodes correspond one-to-one to the columns in A and edges to the rows in A . Without loss of generality, we assume A have m row and n columns and the corresponding graph $G(A)$ has node set $V = \{1, 2, \dots, n\}$. Furthermore, we have the following fact.

Lemma 3. $C = \{i_1, i_2, \dots, i_k\}$ is a CIP set of A if and only if $A(C)$ is a $(3, 2)$ -matrix and $G(A)|_C$ is a graph with maximum degree 2 and $C \subseteq V$.

Assume $A(C)$ has the C1P property. Consider a node $i' \in C$. If it has three adjacent nodes i_j, i_k, i_l . Then, any permutation of C cannot keep the 1's consecutive on the rows corresponding to $(i', i_j), (i', i_k)$ and (i', i_l) since there are only two 1's in each row. Thus, each node in $G(A)|_C$ has degree at most 2. Similarly, we can also show that $G(A)|_C$ does not contains any cycles. Therefore, the node induced subgraph is an union of paths and isolated nodes.

Conversely, if $G(A)|_C$ is an union of paths, then if we arrange all the columns in each path together in the same order as they appear in the path, the resulting matrix have 1's consecutive on each row. This finishes the proof. \square

Using this lemma, we are able to present a 0.5-approximation algorithm for (3, 2)-matrices.

Theorem 5. For any $(3, 2)$ -matrix A with n columns, there exists a $(3, 2)$ -matrix A' with $n/2$ columns such that A' is a 0.5-approximation of A .

Let A be a $(3, 2)$ -matrix. Recall that we assume that each row contains exactly two 1's in A . By Lemma 3, we only need to find a subset C containing at least $n/2$ nodes in $G(A)$ such that the induced subgraph $G(A)|_C$ is an union of paths and isolated nodes. The following ALGORITHM B is such an algorithm.

ALGORITHM B

Input: A $(3, 2)$ -matrix A ;
 Input: A subset C of columns of A such that $A(C)$ has the C1P property.

- 1 Construct the graph $G(A) = (V, E)$ as described above;
 ($G(A)$ has A^t as its incidence matrix; each node has degree at most 3.)
- 2 Initially, set $V' = \phi$ and $V'' = V$;
- 3 Repeat the following action until both $G(A)|_{V'}$ and $G(A)|_{V''}$ contain no nodes of degree more than 1;
- 4 Pick a node of degree at least 2 in $G(A)|_{V'}$ and move it to V'' , or
- 5 Pick a node of degree at least 2 in $G(A)|_{V''}$ and move it to V' ;
- 6 Output $C = V'$ if $|V'| \geq n/2$ or V'' otherwise.

Each execution of line 3-5 increases the number of cut edges between V' and V'' by at least 1; and hence it will repeat at most $|E|$ times. Therefore, the algorithm takes linear time. Since $G(A)|_{V'}$ and $G(A)|_{V''}$ contains no nodes with degree more than 1, by Lemma 3, the output C is a desired subset of columns, i.e. $A(C)$ has the C1P property. This finishes the proof. □

4.2 Generalization to $(\Delta, 2)$ -Matrices

Given a graph G , we use $\Delta(G)$ to denote the largest degree of a node in G . ALGORITHM B indicates that the node set V of any graph G with $\Delta(G) = 3$ can be partitioned into V' and V'' such that $\Delta(G|_{V'}) \leq 1$ and $\Delta(G|_{V''}) \leq 1$. In fact, this is a special case of the following important partition theorem by setting $t_1 = t_2 = 1$.

Theorem 6. Let $G = (V, E)$ be a graph with $\Delta(G) = t_1, t_2, \dots, t_k$ and $\sum_{i=1}^k (t_i + 1) - 1 = \Delta(G)$. Then V can be partitioned into k subsets G_1, G_2, \dots, G_k such that $\Delta(G_i) \leq t_i$ for $i = 1, 2, \dots, k$.

In addition, a desired partition can also be found in polynomial time [13]. By this theorem, the node set of a graph G can be decomposed into $\lceil (\Delta(G) + 1)/2 \rceil$ subsets that induce subgraphs with maximum degree at most 1. This implies the following result.

Proposition 3. Let $G = (V, E)$ be a graph with $\Delta(G) = \Delta$. Then V can be partitioned into $\lceil (\Delta + 1)/2 \rceil$ subsets C such that $|C| \geq n/\lceil (\Delta + 1)/2 \rceil$ and $\Delta(A(C)) \leq 1$.

5 Conclusion

The COS problem finds applications in physical mapping with hybridization data. In this paper, we answer an open problem posed in [12] by proving that the decision version of the COS problem remains NP-complete for $(2, 3)$ and $(3, 2)$ -matrices. To prove these results, we also formulate two simple, but interesting NP-complete problems for cubic graphs. These two problems - Spanning Caterpillar Tree and Induced Disjoint-Path-Union Subgraph may find applications in studying the complexity issues of other algorithmic problems.

We also study the approximation issue of the COS problem. It is proved that the COS problem is 0.8-approximatable for $(2, 3)$ -matrices and 0.5-approximatable for the matrices in which each column contains at most two 1's and for $(3, 2)$ -matrices. But it is open whether the COS problem can be approximatable with constant factor for matrices in which there are at most two 1 in each row.

By studying the complexity and approximation issues of the COS problem that are relevant for physical mapping, we hope our results will give insights into the difficulty of the physical mapping problem which are of value for bioinformaticians.

References

1. J. Atkins and M. Middendorf. On physical mapping and the consecutive ones property for sparse matrices. *Discrete Applied Mathematics*, **71** (1996), 23-40.
2. F. Alizadeh, R. M. Karp, D. K. Weisser and G. Zweig. Physical mapping of chromosomes using unique probes. *Journal of Computational Biology*, **2** (1995), 159-184.
3. K. S. Booth and G. S. Lueker. Test for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Systems Sci.*, **13** (1976), 335-379.
4. J. S. Deogun and K. Gopalakrishnan. Consecutive retrieval property revisited. *Information Processing Letters*, **69** (1999), 15-20.
5. M. Flammini, G. Gambosi and S. Salomone. Boolean routing. *Lecture Notes in Comput. Sci.*, **725** (1993), 219-233.
6. S. Foote, D. Vollrath, A. Hilton and D. C. Page. The human Y chromosome: overlapping DNA clones spanning the euchromatic region. *Science*, **258** (1992), 60-66.
7. D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Mathematics*, **15** (1965), 835-855.
8. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: W. H. Freeman, 1979.
9. S. P. Ghosh. File organization: the consecutive retrieval property. *Commun. ACM*, **15** (1972), 802-808.
10. D. S. Greenberg and S. Istrail. Physical mapping by STS hybridization: algorithmic strategies and the challenge of software evaluation. *J. Comput. Biol.*, **2** (1995), 219-273.
11. M. Habib and R. McConnell, C. Paul and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, **234** (2000), 59-84.
12. M. T. Hajiaghayi and Y. Ganjali. A note on the consecutive ones submatrix problem. *Information Processing Letters*, **83** (2002), 163-166.

13. M. M. Halldórsson and H. C. Lau. Low-degree graph partitioning via local search with applications to constraint satisfaction, max cut, and 3-coloring. *J. Graph Algorithm Appl.*, **1** (3) (1997) 1-13.
14. W.-F. Lu and W.-L. Hsu. A test for the consecutive ones property on noisy data - application to physical mapping and sequence assembly. *Journal of Computational Biology* **10**(5) (2003), 709-735.
15. D. G. Kendall. Incidence matrices, interval graphs and seriation in archaeology. *Pacific J. Math.*, **28** (1969), 565-570.
16. J. M. Lewis. On the complexity of the maximum subgraph problem. in *Proc. 10th Ann. ACM Symp. on Theory of Computing*, (1978) 265-274.
17. L. Lovász. On decomposition of graphs. *Stud. Sci. Math. Hung.*, **1** (1966), 237-238.
18. J. Meidanis, O. Porto and G. P. Telles. On the consecutive ones property. *Discrete Applied Mathematics*, **88** (1998), 325-354.
19. R. Mott, A. Grigoriev, and H. Lehrach. A algorithm to detect chimeric clones and random noise in genomic mapping. *Genetics*, **22** (1994), 482-486.
20. P. A. Pevzner. *Computational molecular biology*. The MIT Press, 2000.
21. S. Weis and R. Reischuk. The complexity of physical mapping with strict chimerism. in *Proc. 6. Int. Symposium on Computing and Combinatorics (COCOON'2000)*, LNCS **1858**, 383-395.
22. M. Yannakakis. Node- and edge-deletion NP-complete problems. in *Proc. 10th Ann. ACM Symp. on Theory of Computing*, (1978) 253-264.

The Two-Guard Problem Revisited and Its Generalization

Xuehou Tan

Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@wing.ncc.u-tokai.ac.jp

Abstract. Given a simple polygon P with two vertices u and v , the *two-guard* problem asks if two guards can move on the boundary chains of P from u to v , one clockwise and one counterclockwise, such that they are mutually visible. By a close study of the structure of the restrictions placed on the motion of two guards, we present a simpler solution to the two-guard problem. The main goal of this paper is to extend the solution for the two-guard problem to that for the *three-guard* problem, in which the first and third guards move on the boundary chains of P from u to v and the second guard is always kept to be visible from them inside P . By introducing the concept of link-2-ray shots, we show a one-to-one correspondence between the structure of the restrictions placed on the motion of two guards and the one placed on the motion of three guards. We can decide if there exists a solution for the three-guard problem in $O(n \log n)$ time, and if so generate a walk in $O(n \log n + m)$ time, where n denotes the number of vertices of P and $m (\leq n^2)$ the size of the optimal walk.

1 Introduction

The *two-guard* problem asks for a walk of two points (called the *guards*) on the boundary of a simple polygon P from the starting vertex u to the ending vertex v , one clockwise and one counterclockwise, such that the guards are always mutually visible. The solution of the two-guard problem consists of several instances of straight walks and counter-straight walks [5]. A walk is said to be *straight* if two guards monotonically move on P from u to v , or *counter-straight* if both guards monotonically move on P clockwise, one from u to v and one from v to u . Icking and Klein gave an $O(n \log n)$ time algorithm for determining if P is straight, counter-straight, or general walkable, where n is the number of vertices of P . Later, a linear time algorithm was presented by Heffernan [4]. Tseng [6] gave an $O(n \log n)$ time algorithm to determine all pairs of boundary points which admits walks, straight walks [7]. Bhattacharya et al. improved this time bound to $O(n)$ [1].

The two-guard problem also involves giving a walk of minimum length, for a (straight) walkable polygon. Icking and Klein have presented an optimal time algorithm for generating a walk of minimum length. To this end, they developed a considerable theorems for evaluating the so-called *hi*, *lo* functions for straight walks, and the *C-hi*, *C-lo* functions for counter-straight walks [5]. It is not a simple work to analyze the performance of these functions.

In this paper, by a close study of the structure of the restrictions placed on the motion of two guards, we present a simpler algorithm to give a walk. For straight walks, we give a method to partition the polygon P into triangular regions so that a sequence of walk instructions can be obtained from the partition, without considering *hi* and *lo* at all. For general walks, we show that counter-straight walks needn't be considered explicitly. Again, the evaluation of *C-hi* and *C-lo* is avoided. This simplification exposes more intrinsic properties of the two-guard problem, and makes our second work easy.

The main goal of this paper is to generalize the solution for the two-guard problem to that for the *visibility graph* problem, in which the first and third guards move on the two boundary chains of P from u to v and the second guard is always kept to be visible from them inside P . By introducing the concept of link-2-ray shots, we show a one-to-one correspondence between the structure of the restrictions placed on the motion of two guards and the one placed on the motion of three guards. We can decide whether there exists a solution for the three-guard problem in $O(n \log n)$ time, and if so generate a walk in $O(n \log n + m)$ time, where $m (\leq n^2)$ denotes the size of the optimal walk. This improves upon the previous time bounds $O(n^2)$ and $O(n^2 \log n)$, respectively [2]. (The corresponding *visibility graph* problem is solved in [2], instead of the *visibility graph* problem.) Moreover, the paradigm developed in this paper is general, and might be used to solve other geometric problems.

2 Preliminaries

Let P be a simple polygon with two marked points u and v . The boundary of P is divided into two chains, L and R , with common endpoints u and v . Both chains L and R are oriented from u to v . For a vertex x of a polygonal chain, $Succ(x)$ denotes the vertex of the chain immediately succeeding x , and $Pred(x)$ the vertex immediately preceding x . For two points $p, p' \in L$, we say that p precedes p' (and $p' \succ p$) if we encounter p before p' when traversing L from u to v . We write $p < p'$ if p precedes p' . The chain $L_{<p}$ (resp. $L_{>p}$) is the subchain of L consisting of all points that precede (resp. succeed) p . For an interval $X \subseteq L$, the point $y \in X$ is the *leftmost* (resp. *rightmost*) if $y \geq x \in X$ (resp. $y \leq x \in X$). We define these concepts for R in a similar manner.

A vertex of P is *reflex* if its interior angle is strictly larger than 180° . The backward ray shot from a reflex vertex r of L or R , denoted by $Backw(r)$, is the first point of P hit by a "bullet" shot at r in the direction from $Succ(r)$ to r , and the forward ray shot $Forw(r)$ is the first point hit by the bullet shot at r in the direction from $Pred(r)$ to r . See Fig. 1. The vertex r is called the *origin* of $Backw(r)$ and $Forw(r)$. Two shots from a chain are said to be *crossing* if the order of them on the opposite chain differs from the order of their origins (e.g., Figs. 1c-d). A pair of vertices $p \in L, q \in R$ is said to give a *crossing* if $q < Backw(p) \in R$ and $p < Backw(q) \in L$ hold (Fig. 1a) or if $q > Forw(p) \in R$ and $p > Forw(q) \in L$ hold (Fig. 1b). A pair of vertices $p, p' \in L$ or $q, q' \in R$ is

said to give a \blacktriangleright if $p < p'$ and $R \ni Forw(p') < Backw(p) \in R$ hold (Fig. 1c) or $q < q'$ and $L \ni Forw(q') < Backw(q) \in L$ hold (Fig. 1d).

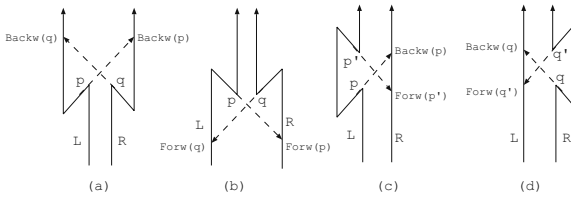


Fig. 1. Deadlocks and wedges

Theorem 1 $\dots P \dots L \dots R \dots P$

In the case that P is walkable, we need to give a \blacktriangleright , which consists of the following actions: (i) both guards move forward along segments of single edges, and (ii) one guard moves forward but the other moves backward [5].

Theorem 2 $\dots O(n) \dots O(n \log n + m) \dots m$

3 The Two-Guard Problem Revisited

Throughout this section, we consider only the ray shots whose chain differs from the chain to which their origins belong. We will write "Forw(p)" or "Backw(p)" to represent "Forw(p) $\in R$ " or "Backw(p) $\in R$ ", and "Forw(q)" or "Backw(q)" to represent "Forw(q) $\in L$ " or "Backw(q) $\in L$ ". We give below our simpler proof of sufficiency of Theorem 1 (the proof of necessity is easy [5]).

3.1 Straight Walks

We give a simple method to show that if L and R are mutually weakly visible and neither deadlocks nor wedges occur, there is a straight walk for P . Our method makes expansive use of Heffernan's idea on the dominance relation defined between ray shots [4]. We define below eight types of \dots (note that the last four types are not defined in [4]). As shown in [4], these dominated shots can be ignored when the straight walk is considered.

- For a vertex $p_2 \in L$, the shot $Backw(p_2)$ (resp. $Forw(p_2)$) is \dots if there exists a vertex $p_1 \in L_{<p_2}$ (resp. $p_1 \in L_{>p_2}$) such that $Backw(p_1) > Backw(p_2)$ (resp. $Forw(p_1) < Forw(p_2)$). See Fig. 2a (resp. Fig. 2b).

- For a vertex $q_2 \in R$, the shot $Backw(q_2)$ (resp. $Forw(q_2)$) is dominated if there exists a vertex $q_1 \in R_{<q_2}$ (resp. $q_1 \in R_{>q_2}$) such that $Backw(q_1) > Backw(q_2)$ (resp. $Forw(q_1) < Forw(q_2)$). See Fig. 2c (resp. Fig. 2d).
- For a vertex $q_2 \in R$, the shot $Forw(q_2)$ (resp. $Backw(q_2)$) is dominated if there exists a vertex $p_1 \in L$ such that $q_2 < Backw(p_1)$ and $p_1 < Forw(q_2)$ (resp. $Forw(p_1) < q_2$ and $Backw(q_2) < p_1$). See Fig. 2e (resp. Fig. 2f).
- For a vertex $p_2 \in L$, the shot $Backw(p_2)$ (resp. $Forw(p_2)$) is dominated if there exists a vertex $q_1 \in R$ such that $Forw(q_1) < p_2$ and $Backw(p_2) < q_1$ (resp. $p_2 < Backw(q_1)$ and $q_1 < Forw(p_2)$). See Fig. 2g (resp. Fig. 2h).

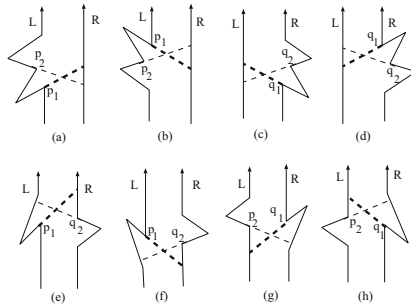


Fig. 2. Illustrating the definition of dominated shots

A shot is said to be *non-dominated* if it is not dominated by any other shots. The key observation we make here is that removing all dominated shots from considerations helps us partition P into triangular regions (Fig. 3b). First, insert the segments connecting four families of non-dominated shots with their origins into P in an arbitrary order. If a segment intersects with some previous or existing segment, it is ignored (i.e., not inserted). This gives a partition of P . See Fig. 3a for an example, where the segments for four families of non-dominated shots are inserted in the order of, backward shots from L , forward shots from L , backward shots from R and forward shots from R .

Let P_i denote a region of the resulting partition, and L_i (resp. R_i) the part of L (resp. R) appearing in P_i (see Fig. 3a). We claim that the whole chain L_i

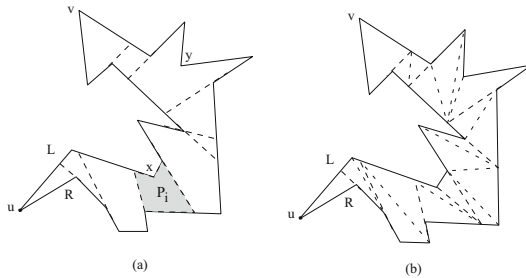


Fig. 3. The decomposition of the polygon P

(resp. R_i) is visible from either endpoint of R_i (resp. L_i). Assume that v is a reflex vertex in P_i . Then, either shot of v is dominated (e.g., $Backw(x)$, $Forw(y)$ and $Backw(y)$ in Fig. 3a), or the segment concerning the shot is not inserted into the partition of P (e.g., $Forw(x)$ in Fig. 3a). Thus, two ray shots from v lie outside of P_i , one above P_i and one below P_i . The vertex v cannot block a point of a chain (L_i or R_i) from being visible from any point of the opposite chain. Our claim is proved. A triangular decomposition of P_i can then be obtained, say, by adding the segments which connect the smallest point of L_i with the vertices of R_i and the largest point of R_i with the vertices of L_i . See Fig. 3b for an example (some triangles have been merged into bigger ones).

Since a sequence of walk instructions can be output from the triangular decomposition described above, we have that if L and R are mutually weakly visible and neither deadlocks nor wedges occur, there is a straight walk for P .

Let us briefly describe the algorithm for determining if P is straight walkable, and if so generating a walk schedule. We call the shots, which are not dominated when only the first four types of dominance relations are considered, "pseudo-non-dominated" shots. It is known that each family of pseudo-non-dominated shots has the non-crossing property [4]. For example, if the origins of pseudo-non-dominated backward shots from L are, in sorted order, p_1, \dots, p_k , then $Backw(p_1), \dots, Backw(p_k)$ are sorted on R . We can compute all pseudo-non-dominated shots, and determine if P is straight walkable, in $O(n)$ time [4].

Our walk schedule can be reported in $O(n)$ time as follows. All non-dominated shots can be computed by merging the set of segments connecting the pseudo-non-dominated backward (resp. forward) shots from L with their origins and the set of segments connecting the pseudo-non-dominated forward (resp. backward) shots with their origins. If any situation shown in Figs. 2e-f (resp. Fig. 2g-h) ever occurs, a dominated shot is found, and its segment is deleted so as to continue the merging procedure. Also, a partition of P into disjoint regions can be obtained by inserting the segments connecting four families of non-dominated shots with their origins into P , using a merging procedure such that whenever a pair of intersecting segments is found, the lately inserted segment is deleted. Finally, each region is divided into triangles, and a sequence of walk instructions is output from the triangular partition of P .

3.2 General Walks

We give a simple method to show that if L and R are mutually weakly visible and no deadlocks occur, there is a walk for the polygon P . As in [5], we define a sequence of maximal wedges in P . (Our definition is the same as that given by Icking and Klein, and all maximal wedges can be computed using their algorithm [5].) Let the 4-tuple $W = \langle a, b, Backw(a), Forw(b) \rangle$ denote a wedge with $a < b$ and $Backw(a) > Forw(b)$. A wedge on L (i.e., $a, b \in L$) is maximal if $Backw(a)$ succeeds all others $Backw(a') \in R$ for $a < a' < b$, $Forw(b)$ precedes all others $Forw(b') \in R$ for $a < b' < b$, and $Forw(b'') < Backw(a'')$ never holds for $a < b'' < a'' < b$. A maximal wedge on R can also be defined. We then have a sequence of maximal wedges in P , ordered from u to v [5].

It is known that the portion of P between two consecutive maximal wedges is straight walkable, and a maximal wedge is counter-straight walkable [5]. The method described in Section 3.1 can be used to give a straight walk. Since the treatment of counter-straight walks given in [5] is complicated, we present below a simpler solution. By symmetry, we discuss only the wedges on L .

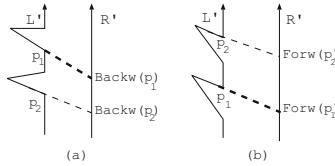


Fig. 4. Illustrating the definition of g -dominated shots

Let L' denote the subchain of L from a to b , and R' the subchain of R from $Forw(b)$ to $Backw(a)$. As in Section 6 of [4], we define the g -dominated shots. (Our definition is the same as that of Heffernan, and all g -dominated shots can be computed using his algorithm [4].) For a vertex $p_2 \in L'$, the shot $Backw(p_2) \in R'$ (resp. $Forw(p_2) \in R'$) is g -dominated if there exists a vertex $p_1 \in L'_{>p_2}$ (resp. $p_1 \in L'_{<p_2}$) such that $Backw(p_1) \in R'$ and $Backw(p_1) > Backw(p_2)$ (resp. $Forw(p_1) \in R'$ and $Forw(p_1) < Forw(p_2)$). See Fig. 4a (resp. Fig. 4b). We then have a list of g -backward (resp. forward) shots such that each shot crosses all others. For an example, if p_1, \dots, p_k are ordered on L' , then $Backw(p_k), \dots, Backw(p_1)$ are ordered on R' .

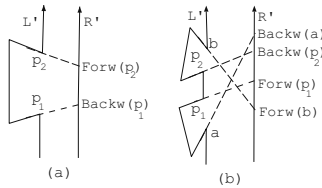


Fig. 5. A possible situation (a); A forbidden situation (b)

Note also that the situation shown in Fig. 5a may occur, but the one shown in Fig. 5b is forbidden (as the considered wedge should end at or before $p_1 Forw(p_1)$).

Lemma 1. *Let L and R be two chains in a walkable structure. Then the set of g -backward shots and the set of g -forward shots are both ordered on R .*

Proof. Let $W = \langle a, b, Backw(a), Forw(b) \rangle$ denote a maximal wedge on L . First, merge the list of non- g -dominated forward shots and the list of non- g -dominated backward shots within W , into a list FB in the decreasing order of them. Beginning with $Backw(a)$, we repeatedly take the shot from FB until all

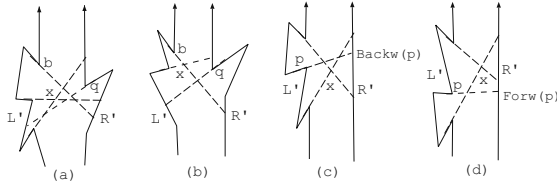


Fig. 6. Illustration for the proof of Lemma 1

shots in FB have been processed. If the next shot does not cross the current shot (e.g., Fig. 5a), it is ignored. In the case that the next shot crosses the current one, we claim that both intervals of L' and R' , determined by two shots and their origins, are visible from the crossing point x . Otherwise, assume there is a reflex vertex $q \in R'$ that blocks a part of R' from being visible from x . In this case, two vertices q and a or b give a deadlock, a contradiction. See Figs. 6a-b for some examples. Assume now there is a reflex vertex $p \in L'$ that blocks a part of L' from being visible from x . If the shot from p crosses both of the considered shots, it is non- g -dominated, a contradiction. See Fig. 6c. Otherwise, one of the considered shots is g -dominated (Fig. 6d), or the situation shown in Fig. 5b (e.g., two considered shots are $Forw(p_1)$ and $Forw(b)$, and the blocking vertex is p_2) occurs, a contradiction. Our claim is proved. The segment connecting two guards can then be rotated around the crossing point x so that two guards move to the next shot and its origin. In this way, W can be walked. \square

4 Generalization to the Three-Guard Problem

Let g_1, g_2 and g_3 denote three guards. A walk schedule consists of the following actions: while g_2 moves along a line segment inside P and is kept visible from g_1 on L and g_3 on R , (i) both g_1 and g_3 move forward along segments of single edges, or (ii) one of g_1 and g_3 moves forward but the other moves backward. If neither backtracking of g_1 on L nor backtracking of g_3 on R is allowed in the walk schedule, then P is said to be straight walkable by three guards.

We will first define the \dots , \dots , and then show a one-to-one correspondence between ray shots and link-2-ray shots so that the results obtained for the two-guard problem can be directly extended to those for the three-guard problem. Most of our geometric arguments are the same as those used in [5].

4.1 Link-2-Ray Shots

Two points $x, y \in P$ are said to be mutually \dots if there exists another point z such that the segments \overline{xz} and \overline{zy} are entirely contained in P . For two regions $Q_1, Q_2 \subseteq P$, we say that Q_1 is \dots from Q_2 if every point in Q_1 is link-2-visible from some point in Q_2 .

Let s denote a segment $pBackw(p)$ (or $p'Forw(p')$) such that if $p \in L$ (or $p' \in L$), then $p < Backw(p) \in L$ (or $p' > Forw(p') \in L$) holds. We call the part

of L between two endpoints of s , the \dots of s . The segment s is said to be \dots if its component contains the component of some other segment. Let S_L denote the set of \dots whose two endpoints belong to L . Similarly, we have the set S_R of non-redundant segments in R .

We give below the definition of link-2-ray shots, which is based on the observation that the shot $Backw(x)$ (resp. $Forw(x)$) is the smallest (resp. largest) point of the opposite chain that is visible from $Succ(x)$ (resp. $Pred(x)$). Suppose that $s = pBackw(p)$ is a segment in S_L and s does not intersect with other segments of S_L . Let $\pi(u, p)$ and $\pi(u, Backw(p))$ denote the shortest paths from u to p and from u to $Backw(p)$, respectively. Let $B(p)$ denote the vertex where two paths diverge. Shoot a "bullet" at $B(p)$ in the direction from the first turn of the path $\pi(B(p), Backw(p))$ to $B(p)$. The hit point on the boundary of P is defined as the \dots from p , denoted by $Backw^2(p)$. See Fig. 7a. Similarly, let $F(p)$ denote the vertex where two paths $\pi(v, p)$ and $\pi(v, Backw(p))$ diverge. Shooting a "bullet" at $F(p)$ in the direction from the first turn of $\pi(F(p), p)$ to $F(p)$ gives the \dots $Forw^2(p)$ (Fig. 7a). Note that the shot $Backw^2(p) \in R$ (resp. $Forw^2(p) \in R$) is the smallest (resp. largest) point of R that is link-2-visible from $Succ(p)$. The link-2-ray shots derived from the segment $pForw(p)$ can be defined analogously (Fig. 7a).

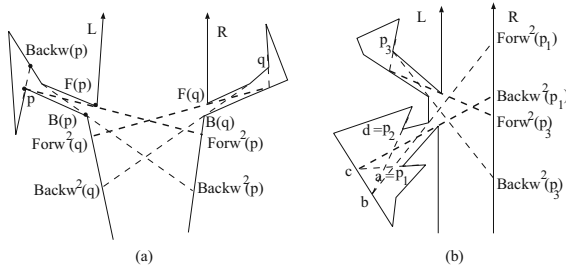


Fig. 7. Link-2-ray shots

Consider now the general case in which the segments of S_L intersect each other. Assume that s_1 is the segment of S_L such that its vertex endpoint p_1 is the smallest in S_L , and s_2, \dots, s_k are the segments of S_L intersecting s_1 , ordered from u to v (Fig. 7b). Let a, b, c, d denote four endpoints of s_1 and s_k , with $a < b < c < d$. See Fig. 7b for an example. We define $Backw^2(p_1)$ and $Forw^2(p_1)$ as those which are computed using the method described above, with a slight modification that p and $Backw(p)$ are replaced by b and c , respectively. The component of any s_i ($1 \leq i \leq k$) contains both b and c , and is thus link-2-visible from $Backw^2(p_1)$ (resp. $Forw^2(p_1)$). Clearly, the shot $Backw^2(p_1) \in R$ (resp. $Forw^2(p_1) \in R$) is the smallest (resp. largest) point on R , from which the component of any s_i is link-2-visible. So $Forw^2(p_1)$ and $Backw^2(p_1)$ can be considered as two link-2-ray shots for this group of intersecting segments (or the vertices). In this way, the link-2-ray shots for the next group of intersecting segments can be defined,

and so on. For the vertices whose non-redundant segments belong to S_R , the link-2-ray shots can be defined analogously.

Similar to the necessary and sufficient condition for the weak visibility between L and R [5], we have the following result (without the proof here).

Lemma 2. $\dots\dots\dots L \dots\dots\dots R \dots\dots\dots$
 $\dots\dots\dots p \in L \dots\dots\dots p < Backw^2(p) \in L \dots\dots\dots p > Forw^2(p) \in L \dots\dots\dots$

Finally, we define the link-2-deadlocks and link-2-wedges. A pair of vertices $p \in L, q \in R$ is said to give a $\dots\dots\dots$ if $Backw^2(p) \in R, Backw^2(q) \in L, q < Backw^2(p)$ and $p < Backw^2(q)$ hold, or $Backw^2(p) \in R, Backw^2(q) \in L, q > Forw^2(p)$ and $p > Forw^2(q)$ hold (Fig. 7a). A pair of vertices $p_1, p_2 \in L$ (resp. $q_1, q_2 \in R$) is said to give a $\dots\dots\dots$ if $Backw^2(p_1) \in R, Forw^2(p_2) \in R, p_1 < p_2$ and $Forw^2(p_2) < Backw^2(p_1)$ hold (resp. if $Backw^2(q_1) \in L, Forw^2(q_2) \in L, q_1 < q_2$ and $Forw^2(q_2) < Backw^2(q_1)$ hold). See Fig. 7b.

4.2 Straight Walkable Case

In this section, we extend the necessary and sufficient condition of straight walks for two guards to that for three guards.

Lemma 3. $\dots\dots\dots P \dots\dots\dots L \dots\dots\dots R \dots\dots\dots$
 $\dots\dots\dots$

Proof. It can be proved by an argument similar to that given for the two-guard problem [5]. We omit the detail in this extended abstract. □

Turn to sufficiency. Recall first that all link-2-ray shots lie on the chain opposite to the chain of their origins. A shot $Backw^2(p_2)$ from a vertex $p_2 \in L$ is $\dots\dots\dots$ if there exists a vertex $p_1 \in L$ such that $p_1 < p_2$ and $Backw^2(p_1) > Backw^2(p_2)$. Similarly, seven other types of dominated link-2-ray shots can be defined. A link-2-ray shot is said to be $\dots\dots\dots$ if it is not dominated by any other shots. Also, each family of non-dominated shots has the $\dots\dots\dots$ property. For example, if $Backw^2(p_1), \dots, Backw^2(p_k)$ are sorted on R , then their "B" points $B(p_1), \dots, B(p_k)$ are sorted on L .

Let *Ray* stand for the symbol "Backw" or "Forw", and *S* the letter "B" or "F". As in Section 3.1, we divide P into disjoint regions by inserting the

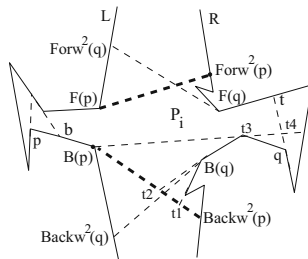
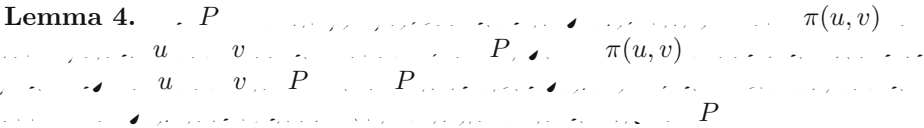


Fig. 8. A region P_i

segments connecting four families of non-dominated shots $Ray^2(z)$ with their points $S(z)$. Let P_i denote a region in the partition of P , and L_i (resp. R_i) the part of L (resp. R) appearing in P_i . By an argument similar to that given in Section 3.1, all the components contained in a chain of P_i are link-2-visible from either endpoint of the opposite chain of P_i . (Note that all points of a chain may not be link-2-visible from an endpoint of the opposite chain.)

Let us see how the region P_i can be walked. Assume that $\overline{S(x)Ray^2(x)}$ (resp. $\overline{S(y)Ray^2(y)}$) is the lower (resp. upper) bounding segment of P_i , and both $Ray^2(x)$ and $Ray^2(y)$ lie on a chain, say, R_i . See Fig. 8. (All other situations can be dealt with analogously.) Let t (resp. b) denote the largest (resp. smallest) point of the components in R_i (resp. L_i). All the components contained in R_i , which are link-2-visible from $S(x)$, can be walked as follows: Fix g_1 at $S(x)$, and start g_2, g_3 at $Ray^2(x)$. Then, move g_3 monotonically to the point t along the chain R_i . At the time that g_3 is going out of the visible region of g_2 , move g_2 to a place such that the edge on which g_3 is moving is visible from it. See Fig. 8 for an example, where $S(x) = B(p)$, $Ray^2(y) = Forw^2(p)$, and the sequence of the points t_i gives the stopped positions of g_2 such that g_1 and g_2 are kept on a line segment as long as possible. Let t_k denote the position of g_2 when g_3 reaches the point t . (In Fig. 8, $t_k = t_4$.) Similarly, all the components in L_i , which are link-2-visible from $Ray^2(y)$, can be walked.

The rest work for walking through P_i is to move g_1 and g_3 on their chains from $S(x)$ to b and from t to $Ray^2(y)$, respectively. Since all the components contained in P_i have been handled, both the chains from $S(x)$ to b and from t to $Ray^2(y)$, are weakly visible from the shortest path $\pi(t_k, Ray^2(y))$ (see Fig. 8). The following result helps complete the walk schedule for P_i .



Proof. Since it is the same as Theorem 7 of [6], the proof is omitted here. \square

By now, we obtain the following results.



Proof. Using the ray-shooting algorithm as well as the shortest paths from u and v to all vertices of P and all end points of segments of S_L and S_R (see Section 4.1), all link-2-ray shots can be calculated in $O(n \log n)$ time. Due to space limit, we omit the detail in this extended abstract. All other steps are similar to those for two guards, and thus take $O(n)$ time. \square

4.3 General Walkable Case

Let us allow backtracking of g_1 and g_3 , i.e., they can move back and forth on the chains L and R . As in Section 3.2, we define a sequence of

link-2-wedges in the polygon P . Let the 4-tuple $W = \langle a, b, \text{Backw}^2(a), \text{Forw}^2(b) \rangle$ denote a link-2-wedge with $a < b$ and $\text{Backw}^2(a) > \text{Forw}^2(b)$. A link-2-wedge W_L on L is maximal if $\text{Backw}^2(a)$ succeeds all others $\text{Backw}^2(a') \in R$ for $a < a' < b$, $\text{Forw}^2(b)$ precedes all others $\text{Forw}^2(b') \in R$ for $a < b' < b$, and $\text{Forw}^2(b'') < \text{Backw}^2(a'')$ never holds for $a < b'' < a'' < b$. A maximal link-2-wedge W_R on R can be defined analogously. Also, we have a sequence of maximal link-2-wedges in P , ordered from u to v . Note that the algorithm for computing maximal wedges [5] can be used to compute maximal link-2-wedges.

We show below that the portion of the polygon P between two consecutive maximal link-2-wedges is straight walkable, and a maximal link-2-wedge can be walked so that g_1 and g_3 move on L and R clockwise.

Lemma 5. Let $W_i = \langle B(a_i), F(b_i), \text{Backw}^2(a_i), \text{Forw}^2(b_i) \rangle$, $W_j = \langle B(a_j), F(b_j), \text{Backw}^2(a_j), \text{Forw}^2(b_j) \rangle$ be consecutive maximal link-2-wedges on L . Then $\overline{B(a_i)F(b_i)} \cap \overline{B(a_j)F(b_j)} = \emptyset$ and $\overline{B(a_i)\text{Backw}^2(a_i)} \cap \overline{B(a_j)\text{Backw}^2(a_j)} = \emptyset$.

Proof. The proof can be given by an argument similar to that for Lemma 5.3 of [5]. We omit the detail in this extended abstract. \square

Consider the walk for a maximal link-2-wedge $W = \langle B(a), F(b), \text{Backw}^2(a), \text{Forw}^2(b) \rangle$ on L . (The walk for a link-2-wedge on R can be given analogously.) Let L' denote the subchain of L from $B(a)$ to $F(b)$, and R' the subchain of R from $\text{Forw}^2(b)$ to $\text{Backw}^2(a)$. For a vertex $p_2 \in L'$, the shot $\text{Backw}^2(p_2) \in R'$ (resp. $\text{Forw}^2(p_2) \in R'$) is g_1 -dominated if there exists a vertex $p_1 \in L'_{>p_2}$ (resp. $p_1 \in L'_{<p_2}$) such that $\text{Backw}^2(p_1) \in R'$ and $\text{Backw}^2(p_1) > \text{Backw}^2(p_2)$ (resp. $\text{Forw}^2(p_1) \in R'$ and $\text{Forw}^2(p_1) < \text{Forw}^2(p_2)$).

Lemma 6. Let $W = \langle B(a), F(b), \text{Backw}^2(a), \text{Forw}^2(b) \rangle$ be a maximal link-2-wedge on L . Then g_1 and g_3 can walk W on L and R clockwise.

Proof. Assume that $W = \langle B(a), F(b), \text{Backw}^2(a), \text{Forw}^2(b) \rangle$ is a maximal link-2-wedge on L . Merge the list of non- g -dominated forward link-2-ray shots and the list of non- g -dominated backward link-2-ray shots, within W , into the list FB in the decreasing order of them. Beginning with $\text{Backw}^2(a)$, we repeatedly take the shot from FB until all shots have been processed. If the next shot does not cross the current shot, it is ignored. Otherwise, we can show by an argument similar to the proof of Lemma 1 that both intervals in L' and R' , determined by two shots and their "S" points, are link-2-visible from the crossing point x of two ray shots. Two guards g_1 and g_3 can then move to the next shot and its "S" point as follows: Move g_2 to the point x , and rotate the segment $\overline{g_1g_3}$ clockwise, using the walk instructions (ii). Whenever the rotation cannot be done, say, g_1 proceeds to a reflex vertex r and is going out of the visible region of g_2 , the guard g_2 moves to the vertex r . Let r' denote the point of L' , which is first touched by

extending of the segment \overline{rx} within P . The subchain of L' from r to r' is weakly visible from the segment $\overline{rr'}$; otherwise, the chain L' is not link-2-visible from x , a contradiction. Then, we monotonically move g_1 to r' along his chain and move g_2 to r' along the segment $\overline{rr'}$ (see also Theorem 5 of [6]). Finally, move g_2 back to the point x . In this way, the wedge W can be walked. The number of walk instructions used is linear in the size of W . \square

By now, we can conclude the main result of this paper.

Theorem 5. *Let P be a simple polygon with n vertices and m link-2-wedges. Then, the sequence of maximal link-2-wedges in P can be computed in $O(n \log n)$ time.*

Theorem 6. *Let P be a simple polygon with n vertices and m link-2-wedges. Then, the sequence of maximal link-2-wedges in P can be walked in $O(n \log n + m)$ time.*

Proof. First, all link-2-ray shots are computed in $O(n \log n)$ time. Similar to the argument used in [4, 5], the walkability of the polygon can then be determined in $O(n)$ time. Using the algorithm in [5], we compute the sequence of maximal link-2-wedges in P . As in [5], we can show that these wedges give the necessary and sufficient turning points for g_1 and g_3 . Applying the result of Section 4.2 for the region between two maximal link-2-wedges and Lemma 6 for a link-2-wedge gives a general walk. The size of our walk is clearly $O(m)$. \square

Lemma 7. *Let P be a simple polygon with n vertices and m link-2-wedges. Then, the number of maximal link-2-wedges in P is $\Theta(n^2)$.*

Proof. Fig. 13 of [2] shows a case in which $m = \Omega(n^2)$. On the other hand, since there are $O(n)$ maximal link-2-wedges, the number of the walks for them (Lemma 6) and the straight walks between them is $O(n)$. Since each of these walks takes $O(n)$ time, the number of walk instructions used is $O(n^2)$. \square

Finally, the paradigm developed in this paper can be used to solve the general k -guard problem. Such a result should have an application in the problem of sweeping through a simple polygon with a chain of minimum number of guards, without considering any starting or ending point [3].

References

- [1] B.K.Bhattacharya, A. Mukhopadhyay and G.Narasimhan, "Optimal algorithms for two-guard walkability of simple polygons", *Lect. Notes Comput. Sci.* **2125** (2001) 438-449.
- [2] D.Crass, I.Suzuki and M.Yamashita, "Search for a mobile intruder in a corridor", *IJCGA* **5** (1995) 397-412.
- [3] A.Efrat, L.J.Guibas, S. Har-Peled, D.C.Lin, J.S.B. Mitchell and T.M.Murali, "Sweeping simple polygons with a chain of guards", In *Proc., ACM-SIAM SODA* (2000) 927-936.
- [4] P.J.Heffernan, "An optimal algorithm for the two-guard problem", *IJCGA* **6** (1996) 15-44.
- [5] C. Icking and R. Klein, "The two guards problem", *IJCGA* **2** (1992) 257-285.
- [6] I.Suzuki and M.Yamashita, "Searching for mobile intruders in a polygonal region", *SIAM J. Comp.* **21** (1992) 863-888.
- [7] L.H.Tseng, P.J.Heffernan and D.T.Lee, "Two-guard walkability of simple polygons", *IJCGA* **8** (1998) 85-116.

Canonical Data Structure for Interval Probe Graphs

Ryuhei Uehara

Department of Information Processing, School of Information Science, JAIST, Ishikawa, Japan
uehara@jaist.ac.jp

Abstract. The class of interval probe graphs is introduced to deal with the physical mapping and sequencing of DNA as a generalization of interval graphs. The polynomial time recognition algorithms for the graph class are known. However, the complexity of the graph isomorphism problem for the class is still unknown. In this paper, extended \mathcal{MPQ} -trees are proposed to represent the interval probe graphs. An extended \mathcal{MPQ} -tree is canonical and represents all possible permutations of the intervals. The extended \mathcal{MPQ} -tree can be constructed from a given interval probe graph in polynomial time. Thus we can solve the graph isomorphism problem for the interval probe graphs in polynomial time. Using the tree, we can determine that any two nonprobes are independent, overlapping, or their relation cannot be determined without an experiment. Therefore, we can heuristically find the best nonprobe that would be probed in the next experiment. Also, we can enumerate all possible affirmative interval graphs for any interval probe graph.

Keywords: Bioinformatics, data structure, graph isomorphism, interval probe graph.

1 Introduction

The class of interval graphs was introduced in the 1950's by Hajös and Benzer independently. Since then a number of interesting applications for interval graphs have been found including to model the topological structure of the DNA molecule, scheduling, and others (see [6, 14, 4] for further details). The interval graph model requires all overlap information. However, in many cases, only partial overlap data exist. The class of interval probe graphs is introduced by Zhang in the assembly of contigs in physical mapping of DNA, which is a problem arising in the sequencing of DNA (see [19, 21, 20, 14] for background). An interval probe graph is obtained from an interval graph by designating a subset P of vertices as *probes*, and removing the edges between pairs of vertices in the remaining set N of *nonprobes*. That is, on the model, only partial overlap information (between a probe and the others) is given. From the graph theoretical point of view, interval probe graphs are related to tolerance graphs [8–Section 4], and recently, the notion is extended to the chordal probe graphs [7, 16]. On the other hand, from the practical point of view, a few efficient algorithms for the class are known; the recognition algorithms [10, 13, 9], and an algorithm for finding a tree 7-spanner (see [3] for details). The recognition algorithm in [10] also gives a data structure that represents all possible permutations of the intervals of an interval probe graph.

A data structure called \mathcal{PQ} -trees was developed by Booth and Lueker to represent all possible permutations of the intervals of an interval graph [2]. Korte and Möhring

simplified their algorithm by introducing \mathcal{MPQ} -trees [11]. An \mathcal{MPQ} -tree is canonical; that is, given two interval graphs are isomorphic if and only if their corresponding \mathcal{MPQ} -trees are isomorphic. However, there are no canonical \mathcal{MPQ} -trees for interval probe graphs. In general, given an interval probe graph, there are several affirmative interval graphs those are not isomorphic, and their interval representations are consistent to the interval probe graph.

In this paper, we extend \mathcal{MPQ} -trees to represent interval probe graphs. The extended \mathcal{MPQ} -tree is canonical for any interval probe graph, and the tree can be constructed in polynomial time. Thus the graph isomorphism problem for interval probe graphs can be solved in polynomial time. From the theoretical point of view, the complexity of the graph isomorphism of interval probe graphs was not known (see [18] for related results and references). Thus the result improves the upper bound of the graph classes such that the graph isomorphism problem can be solved in polynomial time.

From the practical point of view, the extended \mathcal{MPQ} -tree is very informative, which is beneficial in the Computational Biology community. The extended \mathcal{MPQ} -tree gives us the information between nonprobes; the relation of two nonprobes is either (1) independent (they cannot overlap with each other), (2) overlapping, or (3) not determined without experiments. Hence, to clarify the structure of the DNA sequence, we only have to experiment on the nonprobes in the case (3). Moreover, given extended \mathcal{MPQ} -tree, we can find the nonprobe v that has the most nonprobes u such that v and u are in the case (3) in linear time. Therefore, we can heuristically find the “best” nonprobe to fix the structure of the DNA sequence efficiently.

Due to space limitation, details of the construction of an extended \mathcal{MPQ} -tree are omitted.¹

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$, and the *degree* of a vertex v is $|N_G(v)|$ and denoted by $\deg_G(v)$. For the vertex set U of V , we denote by $N_G(U)$ the set $\{v \in V \mid v \in N(u) \text{ for some } u \in U\}$. If no confusion can arise we will omit the index G . Given a graph $G = (V, E)$ and a subset $U \subseteq V$, the *subgraph of G induced by U* is the graph (U, F) , where $F = \{\{u, v\} \mid \{u, v\} \in E \text{ for } u, v \in U\}$, and denoted by $G[U]$. Given a graph $G = (V, E)$, its *complement* $\bar{G} = (V, \bar{E})$ is defined by $\bar{E} = \{\{u, v\} \mid u, v \in V \text{ and } \{u, v\} \notin E\}$. A vertex set I is *independent set* if $G[I]$ contains no edges, and then the graph $\bar{G}[I]$ is said to be a *clique*.

For a given graph $G = (V, E)$, a sequence of the distinct vertices v_0, v_1, \dots, v_l is a *path*, denoted by (v_0, v_1, \dots, v_l) , if $\{v_j, v_{j+1}\} \in E$ for each $0 \leq j \leq l - 1$. The *length* of a path is the number of edges on the path. For two vertices u and v , the *distance* of the vertices is the minimum length of the paths joining u and v . A *cycle* is a path beginning and ending with the same vertex. An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of that cycle. A graph is *chordal* if each cycle of

¹ Full draft is available at <http://www.komazawa-u.ac.jp/~uehara/ps/MPQipg.pdf>

length at least 4 has a chord. Given graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $G[N(v)]$ is a clique in G . The following lemma is a folklore (see [17]):

Lemma 1. *Given chordal graph, all simplicial vertices can be found in linear time.*

The ordering v_1, \dots, v_n of the vertices of V is a *perfect elimination ordering* of G if the vertex v_i is simplicial in $G[\{v_i, v_{i+1}, \dots, v_n\}]$ for all $i = 1, \dots, n$. Then a graph is chordal if and only if it has a perfect elimination ordering (see, e.g., [4–Section 1.2] for further details).

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if and only if there is a one-to-one mapping $\phi : V \rightarrow V'$ which satisfies $\{u, v\} \in E$ if and only if $\{\phi(u), \phi(v)\} \in E'$ for every pair of vertices u and v . We denote by $G \sim G'$ if G is isomorphic to G' . Given graphs G and G' , *graph isomorphism problem* is the problem to determine if $G \sim G'$.

2.1 Interval Graph Representation

A graph (V, E) with $V = \{v_1, v_2, \dots, v_n\}$ is an *interval graph* if there is a set of intervals $\mathcal{I} = \{I_{v_1}, I_{v_2}, \dots, I_{v_n}\}$ such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $1 \leq i, j \leq n$. We call the set \mathcal{I} of intervals *interval representation* of the graph. For each interval I , we denote by $R(I)$ and $L(I)$ the right and left endpoints of the interval, respectively (hence we have $L(I) \leq R(I)$ and $I = [L(I), R(I)]$).

A graph $G = (V, E)$ is an *interval probe graph* if V can be partitioned into subsets P and N (corresponding to the *probes* and *nonprobes*) and each $v \in V$ can be assigned to an interval I_v such that $\{u, v\} \in E$ if and only if both $I_u \cap I_v \neq \emptyset$ and at least one of u and v is in P . In this paper, we assume that P and N are given, and then we denote by $G = (P, N, E)$. By definition, N is an independent set, and $G[P]$ is an interval graph. Let $G = (P, N, E)$ be an interval probe graph. Let E^+ be a set of edges $\{t_1, t_2\}$ with $t_1, t_2 \in N$ such that there are two probes v_1 and v_2 in P such that $\{v_1, t_1\}, \{v_1, t_2\}, \{v_2, t_1\}, \{v_2, t_2\} \in E$, and $\{v_1, v_2\} \notin E$. In the case, we can know that intervals t_1 and t_2 have to overlap without experiment. Each edge in E^+ is called an *enhanced edge*, and the graph $G^+ := (P, N, E \cup E^+)$ is said to be an *enhanced interval probe graph*. It is known that an interval probe graph is weakly chordal [15], and an enhanced interval probe graph is chordal [19, 21]. For further details and references can be found in [4, 14].

For a given (enhanced) interval probe graph G , an interval graph G' is said to be *affirmative* if and only if G' gives one possible interval representations for G . In general, for an (enhanced) interval probe graph G , there are several non-isomorphic affirmative interval graphs.

2.2 PQ-Trees and MPQ-Trees

PQ -trees were introduced by Booth and Lueker [2], and which can be used to recognize interval graphs as follows. 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 are labeled 1-1 with the maximal cliques of the interval graph G . The *frontier* of a PQ -tree T is the permutation of the maximal cliques obtained by the ordering of the leaves

of T from left to right. \mathcal{PQ} -tree T and T' are *equivalent*, if one can be obtained from the other by applying the following rules a finite number of times; (1) arbitrarily permute the successor nodes of a \mathcal{P} -node, or (2) reverse the order of the successor nodes of a \mathcal{Q} -node. In [2], Booth and Lueker showed that a graph G is an interval graph if and only if there is a \mathcal{PQ} -tree T whose frontier represents a consecutive arrangement of the maximal cliques of G . They also developed a linear algorithm that either constructs a \mathcal{PQ} -tree for G , or states that G is not an interval graph. If G is an interval graph, then all consecutive arrangements of the maximal cliques of G are obtained by taking equivalent \mathcal{PQ} -trees.

Lueker and Booth [12], and Colbourn and Booth [5] developed labeled \mathcal{PQ} -trees in which each node contains information of vertices as labels. Their labeled \mathcal{PQ} -trees are *canonical*; given interval graphs G_1 and G_2 are isomorphic if and only if corresponding labeled \mathcal{PQ} -trees T_1 and T_2 are isomorphic. Since we can determine if two labeled \mathcal{PQ} -trees T_1 and T_2 are isomorphic, the isomorphism of interval graphs can be determined in linear time.

MPQ -trees, which stands for *modified PQ-trees*, are developed by Korte and Möhring to simplify the construction of \mathcal{PQ} -trees [11]. The MPQ -tree T^* assigns sets of vertices (possibly empty) to the nodes of a \mathcal{PQ} -tree T representing an interval graph $G = (V, E)$. A \mathcal{P} -node is assigned only one set, while a \mathcal{Q} -node has a set for each of its sons (ordered from left to right according to the ordering of the sons). A \mathcal{P} -node \hat{P} consists of those vertices of G contained in all maximal cliques represented by the subtree or \hat{P} in T , but in no other cliques.² The definition of a \mathcal{Q} -node \hat{Q} is more involved: Let Q_1, \dots, Q_m ($m \geq 3$) be the set of the sons (in consecutive order) of \hat{Q} , and let T_i be the subtree of T with root Q_i . We then assign a set S_i , called *section*, to \hat{Q} for each Q_i . Section S_i contains all vertices that are contained in all maximal cliques of T_i and some other T_j , but not in any clique belonging to some other subtree of T that is not below \hat{Q} .

In [11], Korte and Möhring proposed two linear time algorithms that construct an MPQ -tree for given interval graph. Although it does not shown explicitly, the MPQ -trees constructed by the algorithms are the same. The MPQ -tree directly corresponds to the labeled \mathcal{PQ} -tree; the sets of vertices assigned in the MPQ -tree directly correspond to the “characteristic nodes” in [5]. Hence, the labeled \mathcal{PQ} -tree is canonical, so is the MPQ -tree. Therefore the graph isomorphism problem also can be solved in linear time using the MPQ -trees (without constructing \mathcal{PQ} -trees in [2]). The property of MPQ -trees is summarized as follows [11–Theorem 2.1]:

Theorem 1. *Let T^* be the canonical MPQ -tree for an interval graph $G = (V, E)$. Then*

- (a) T^* can be obtained in $O(|V| + |E|)$ time and $O(|V|)$ space.
- (b) Each maximal clique of G corresponds to a path in T^* from the root to a leaf, where each vertex $v \in V$ is as close as possible to the root.
- (c) In T^* , each vertex v appears in either one leaf, one \mathcal{P} -node, or consecutive sections $S_i, S_{i+1}, \dots, S_{i+j}$ (with $j > 0$) in a \mathcal{Q} -node.
- (d) The root of T^* contains all vertices belonging to all maximal cliques, while the leaves contain the simplicial vertices.

² We will use \hat{P} , \hat{Q} , and \hat{N} for describing a \mathcal{P} -node, \mathcal{Q} -node, any node, respectively to distinguish probe set P and nonprobe set N .

Proof. The claims (a) and (b) are stated in [11–Theorem 2.1]. The claim (c) is immediately obtained by the fact that the maximal cliques containing a fixed vertex occur consecutively in T ; see [11] for further details. The claim (d) is also stated in [11–p. 71]. \square

We note that there are no vertices that appear in only one section in a \mathcal{Q} -node (since such a vertex appears in the subtree of the section). Thus each vertex in a \mathcal{Q} -node appears at least two consecutive sections.

Lemma 2. *Let \hat{Q} be a \mathcal{Q} -node in the canonical \mathcal{MPQ} -tree. Let S_1, \dots, S_k (in this order) be the sections of \hat{Q} , and let U_i denote the set of vertices occurring below S_i with $1 \leq i \leq k$. Then we have the following;*

- (a) $S_{i-1} \cap S_i \neq \emptyset$ for $2 \leq i \leq k$,
- (b) $S_1 \subseteq S_2$ and $S_k \subseteq S_{k-1}$,
- (c) $U_1 \neq \emptyset$ and $U_k \neq \emptyset$,
- (d) $(S_i \cap S_{i+1}) \setminus S_1 \neq \emptyset$ and $(S_{i-1} \cap S_i) \setminus S_k \neq \emptyset$ for $2 \leq i \leq k-1$,
- (e) $S_{i-1} \neq S_i$ with $2 \leq i \leq k-1$, and
- (f) $(S_{i-1} \cup U_{i-1}) \setminus S_i \neq \emptyset$ and $(S_i \cup U_i) \setminus S_{i-1} \neq \emptyset$ for $2 \leq i \leq k$.

Proof. The results in [11] lead us from (a) to (e) immediately. Thus we show (f). If $(S_{i-1} \cup U_{i-1}) \setminus S_i = \emptyset$, we have $U_{i-1} = \emptyset$ and $S_{i-1} \subset S_i$. In the case, S_{i-1} is redundant section; we can obtain more compact \mathcal{MPQ} -tree by removing S_{i-1} . This fact contradicts that the \mathcal{MPQ} -tree is canonical. Thus (f) is settled. \square

For an enhanced interval probe graph $G^+ = (P, N, E \cup E^+)$, let u and v be any two nonprobes with $\{u, v\} \notin E^+$. Then, we say that u intersects v if $I_u \cap I_v \neq \emptyset$ for all affirmative interval graphs of G^+ . The nonprobes u and v are *independent* if $I_u \cap I_v = \emptyset$ for all affirmative interval graphs of G^+ . Otherwise, we say that the nonprobe u *potentially intersects* v . Intuitively, if u potentially intersects v , we cannot determine their relation without experiments.

2.3 Extended \mathcal{MPQ} -Trees

If given graph is an interval graph, the corresponding \mathcal{MPQ} -tree is uniquely determined up to isomorphism. However, for an interval probe graph, this is not in the case. For example, consider an interval probe graph $G = (P, N, E)$ with $P = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $N = \{a, b, c, d, e, f, g\}$ given in Fig. 1. If the graph does not contain the nonprobe g , we have the canonical \mathcal{MPQ} -tree in Fig. 2. However, the graph is an interval probe graph and we do not know if g intersects b and/or c since they are nonprobes. According to the relations between g and b and/or c , we have four possible \mathcal{MPQ} -trees that are affirmative to G shown in Fig. 3, where X is either $\{1, 2, 7, 8\}$, $\{1, 2, 7, 8, c\}$, or $\{1, 2, 7, 8, b, c\}$. We call such a vertex g *floating leaf* (later, it will be shown that such a vertex has to be a leaf in an \mathcal{MPQ} -tree). For a floating leaf, there is a corresponding \mathcal{Q} -node (which also will be shown later). Thus we extend the notion of a \mathcal{Q} -node to contain the information of the floating leaf. A floating leaf appears consecutive sections of a \mathcal{Q} -node \hat{Q} as the ordinary vertices in \hat{Q} . To distinguish them, we draw them over the corresponding sections; see Fig. 4. Further details will be discussed in Section 3.

Note that the tree constructed in step A2 is the *ordinary* \mathcal{MPQ} -tree. In step A3, it will be modified to the extended \mathcal{MPQ} -tree.

3.1 Property of G^*

Let $G^* = (P, N^*, E^*)$ be the enhanced interval probe graph induced by P and N^* in step A2. The following lemma plays an important role in this subsection.

Lemma 3. *Let u and v be any nonprobes in N^* . Then there is an interval representation of G^* such that $I_u \cap I_v \neq \emptyset$ if and only if $\{u, v\} \in E^+$.*

Proof. If $\{u, v\} \in E^+$, $I_u \cap I_v \neq \emptyset$ by definition. Thus we assume that $\{u, v\} \notin E^+$, and show that there is an interval representation of G^* such that $I_u \cap I_v = \emptyset$. We fix an interval representation of G^* , and assume that $I_u \cap I_v \neq \emptyset$. When $N(u) \cap N(v) = \emptyset$, it is easy to modify to satisfy $I_u \cap I_v = \emptyset$. Thus we assume that $N(u) \cap N(v) \neq \emptyset$. We first show that $N(u) \not\subseteq N(v)$ and $N(v) \not\subseteq N(u)$. If $N(u) \subseteq N(v)$, since $\{u, v\} \notin E^+$, all vertices in $N(u)$ intersect with each other. Thus, $N(u)$ induces a clique, which contradicts $u \in N^*$. Hence $N(u) \not\subseteq N(v)$ and $N(v) \not\subseteq N(u)$. Without loss of generality, we can assume that $L(u) < L(v) < R(u) < R(v)$. Let w_1 and w_2 are any probes which intersect the interval $[L(v), R(u)]$. Then, since $\{u, v\} \notin E^+$, $I_{w_1} \cap I_{w_2} \neq \emptyset$. Thus, by the Helly property (see, e.g., [1]), there is a point p in the interval $[L(v), R(u)]$ such that all probes contain p . We replace the point p in all intervals by a small interval $[p - \epsilon, p + \epsilon]$, and then we replace I_u by $[L(u), p - \epsilon]$ and I_v by $[p + \epsilon, R(v)]$. The replacement has no effect to the relations between u (or v) and probes. We here show that the replacement also has no effect to the relations between u (or v) and other nonprobes. To derive contradictions, we assume that the relation between u and a nonprobe w is changed. Since the interval I_u is shortened, $w \in N(u)$ becomes $w \notin N(u)$ by the replacement. Since both of u and w are nonprobes, there are two independent probes t_1 and t_2 that guarantee $w \in N(u)$. Then, replacing $[L(u), R(u)]$ by $[L(u), p - \epsilon]$, at least one of t_1 and t_2 , say t , changes from $t \in N(u)$ to $t \notin N(u)$. However this contradicts the definition of the point p , which should be contained in t , and we have $t \in N(u)$ after replacement. Thus the replacement has no effect to the relations between u (or v) and other nonprobes. Hence we obtain a new affirmative interval representation of G^* with $I_u \cap I_v = \emptyset$. Repeating this process for each pair we have the lemma. □

The definition of (enhanced) interval probe graphs and Lemma 3 imply the main theorem in this subsection:

Theorem 2. *The enhanced interval probe graph G^* is an interval graph.*

Hereafter we call the graph $G^* = (P, N^*, E^*)$ the *backbone interval graph* of $G^+ = (P, N, E \cup E^+)$. For any given interval graph, its corresponding \mathcal{MPQ} -tree can be computed in linear time [11]. Thus we also have the following corollary:

Corollary 1. *The canonical \mathcal{MPQ} -tree T^* of G^* can be computed in linear time.*

In the canonical \mathcal{MPQ} -tree T^* , for each pair of nonprobes u and v , their corresponding intervals intersect if and only if $\{u, v\} \in E^+$. This implies the following observation.

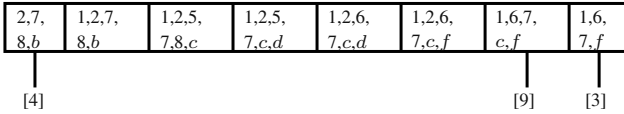


Fig. 5. The canonical MPQ-tree T^* of G^*

Observation 1. *The canonical MPQ-tree T^* gives us the possible interval representations of G^* such that two nonprobes in N^* do not intersect as possible as they can.*

For example, for the graph $G = (P, N, E)$ in Fig. 1, the canonical MPQ-tree of the backbone interval graph $G^* = (P, N^*, E^*)$ is described in Fig. 5. In the MPQ-tree, $I_d \cap I_f = \emptyset$.

Now, our main task is that embedding each vertex in N_S into the canonical MPQ-tree T^* without breaking canonicity.

3.2 Embedding of Nonprobes in N_S

Due to space limitation, we describe an outline of our embedding.

We first show two lemmas for the nonprobes in N_S .

Lemma 4. *For each nonprobe v in N_S , all vertices in $N(v)$ are probes.*

Proof. To derive a contradiction, we assume that a nonprobe v' is in $N(v)$. Then $\{v, v'\}$ is in E^+ . Thus there are two probes u and u' such that $\{u, v\}$, $\{u, v'\}$, $\{u', v\}$, and $\{u', v'\}$ are in E , and $\{u, u'\}$ is not in E , which contradicts that $v \in N_S$. \square

Lemma 5. *For any interval probe graph G , there is an affirmative interval graph G' such that every nonprobe v in N_S of G is also simplicial in G' .*

Proof. Let v be any nonprobe in N_S such that v is not simplicial in G' . By the Helly property, there is a point p such that all probes in $N(v)$ contains p . We replace the point p in all intervals by a small interval $[p - \epsilon, p + \epsilon]$, and we set $R(v) = L(v) = p$. Then v is simplicial in the interval graph corresponding to the new interval representation, and the interval graph is still affirmative. Thus, repeating this process, we have the lemma. \square

By Lemma 5 and Theorem 1(d), we have the following corollary.

Corollary 2. *For any interval probe graph G , there is an affirmative interval graph G' such that every nonprobe v in N_S of G is in a leaf of the MPQ-tree of G' .*

We will construct the extended MPQ-tree that represents all such affirmative interval graphs stated in Corollary 2 for G .

Our embedding is an extension of the embedding by Korte and Möhring [11] to deal with nonprobes. Hereafter, we suppose that the algorithm picks up some nonprobe v from N_S and it is going to embed v into T^* . Each node \hat{N} (including Q -node) of the current tree T^* and each section S of a Q -node is labeled according to how the nonprobe v in N_S is related to the probes in \hat{N} or S . Nonprobes in \hat{N} or S are ignored. The label is ∞ , 1, or 0 if v is adjacent to all, some, or no probe from \hat{N} , or S , respectively. Empty sets (or the sets containing only nonprobes) obtain the label 0. Labels 1 and ∞ are called *positive labels*.

Lemma 6. *For a nonprobe v in N_S , all nodes with a positive label are contained in a unique path of T^* .*

Proof. By definition, v is simplicial, or $N(v)$ induce a clique. Thus Theorem 1(b) implies the lemma. □

Let P' be the unique minimal path in T^* containing all nodes with positive label. Let P be a path from the root of the \mathcal{MPQ} -tree T^* to a leaf containing P' (a leaf is chosen in any way). Let \hat{N}_* be the lowest node in P with positive label. If P contains nonempty \mathcal{P} -nodes or sections above \hat{N}_* with label 0 or 1, let \hat{N}^* be the highest such \mathcal{P} -node or \mathcal{Q} -node containing the section. Otherwise put $\hat{N}_* = \hat{N}^*$. Intuitively, the neighbors of v are distributed in the nodes from \hat{N}_* to \hat{N}^* along P .

When $\hat{N}_* \neq \hat{N}^*$, we have the following lemma:

Lemma 7. *We assume that $\hat{N}_* \neq \hat{N}^*$. Let \hat{Q} be any \mathcal{Q} -node with sections S_1, \dots, S_k in this order between \hat{N}_* and \hat{N}^* . If \hat{Q} is not \hat{N}^* , all neighbors of v in \hat{Q} appear in either S_1 or S_k .*

Proof. We first observe that \hat{N}^* contains at least one probe w of v with $w \notin N(v)$ since \hat{N}^* is non-empty and the label of \hat{N}^* is 0 or 1. We assume that v has a neighbor u in \hat{Q} with $u \notin S_1$ and $u \notin S_k$ to derive a contradiction. Let U_1 and U_k be the set of vertices occurring below S_1 and S_k , respectively. By Lemma 2(c), $U_1 \neq \emptyset$ and $U_k \neq \emptyset$. Thus there are two vertices $u_1 \in U_1$ and $u_k \in U_k$ such that $I_{u_1} \subseteq I_w, I_u \subseteq I_w, I_{u_k} \subseteq I_w$, and $R(I_{u_1}) < L(I_u) < R(I_u) < L(u_k)$ (or $R(I_{u_k}) < L(I_u) < R(I_u) < L(u_1)$). Thus we have $I_u \subset I_w$, which contradicts that $w \notin N(v)$ and $u \in N(v)$. □

We note that Lemma 7 does not hold at the node \hat{N}^* .

We are now ready to use the bottom-up strategy from \hat{N}_* to \hat{N}^* as in [11]. In [11], the ordering of vertices are determined by LexBFS. In our algorithm, the step A3 consists of the following substeps;

- A3.1. while there is a nonprobe v such that $\hat{N}_* \neq \hat{N}^*$ for v , embed v into T^* ;
- A3.2. while there is a nonprobe v such that $\hat{N}_* = \hat{N}^*$ for v and v is not a floating leaf, embed v into T^* ;
- A3.3. embed each nonprobe v (such that $\hat{N}_* = \hat{N}^*$ for v and v is a floating leaf) into T^* .

An embedding of a nonprobe v with $\hat{N}_* \neq \hat{N}^*$ merges some nodes into one new \mathcal{Q} -node. Thus, during step A3.1, embedding of a nonprobe v can change the condition of other nonprobes u from “ $\hat{N}_* \neq \hat{N}^*$ ” to “ $\hat{N}_* = \hat{N}^*$ ”.

We here have the following theorem:

Theorem 3. *When $\hat{N}_* \neq \hat{N}^*$, v is not a floating leaf.*

The proof of Theorem 3 is omitted here since it hardly depends on the templates for the embedding process. By Theorem 3, steps A3.1 and A3.2 do not generate floating leaves, and all floating leaves are embedded in step A3.3. Hence the templates used in steps A3.1 and A3.2 are not required to manage floating leaves.

When $\hat{N}_* = \hat{N}^*$, the embedding is done at the unique node $\hat{N}_* = \hat{N}^*$. On the other hand, when $\hat{N}_* \neq \hat{N}^*$, the embedding is involved. But the number of cases are decreased by Lemma 7: Our embedding from \hat{N}_* to \hat{N}^* along P is the natural extension of the construction of the \mathcal{MPQ} -tree in [11] except at the top node \hat{N}^* . At the node \hat{N}^* , our embedding is natural extension of the construction of the \mathcal{PQ} -tree in [2].

Example 1. For the graph $G = (P, N, E)$ in Fig. 1 with its backbone interval graph in Fig. 5, the extended \mathcal{MPQ} -tree \tilde{T} is shown in Fig. 4. Note that we can know that e intersects both of c and d with neither experiments nor enhanced edges. We also note that I_a and I_b could have intersection, but they are standardized.

3.3 Analysis of Algorithm

The correctness and complexity of the algorithm is given by the straightforward (but tedious) case analysis for the templates. Hence the proof is omitted here.

Theorem 4. *The resultant extended \mathcal{MPQ} -tree is canonical up to isomorphism.*

Theorem 5. *For a given interval probe graph $G = (P, N, E)$, let \tilde{T} be the canonical extended \mathcal{MPQ} -tree, and $G^+ = (P, N, E \cup E^+)$ be the corresponding enhanced interval graph. Let \tilde{E} be the set of edges $\{v_1, v_2\}$ joining nonprobes v_1 and v_2 which is given by \tilde{T} ; more precisely, we regard \tilde{T} as the ordinary \mathcal{MPQ} -tree, and the graph $\tilde{G} = (P \cup N, E \cup E^+ \cup \tilde{E})$ is the interval graph given by the \mathcal{MPQ} -tree \tilde{T} (thus a floating leaf may not be a leaf). Then \tilde{T} can be computed in $O((|P| + |N|)|E| + |E^+| + |\tilde{E}|)$ time and $O(|P| + |N| + |E| + |E^+| + |\tilde{E}|)$ space.*

Corollary 3. *The graph isomorphism problem for the class of (enhanced) interval probe graphs G is solvable in $O(n^2 + nm)$ time and $O(n^2)$ space, where n and m are the number of vertices and edges of an affirmative interval graph of G , respectively.*

We note that $|E^+| + |\tilde{E}|$ can be $\Theta(|N|^2) = \Theta(n^2)$ even if $|E| = O(n)$, where $n = |P| + |N|$. Thus the running time in the main theorem can be $\Theta(n^3)$ even if given interval probe graphs have $O(n)$ edges.

4 Applications

Given canonical extended \mathcal{MPQ} -tree \tilde{T} , using a standard depth first search technique, we can compute in linear time if each subtree in \tilde{T} contains only nonprobes. Thus, hereafter, we assume that each section S_i knows if its subtree contains only nonprobes or not.

We first consider the following problem:

Input: An enhanced interval probe graph $G^+ = (P, N, E \cup E^+)$ and the canonical extended \mathcal{MPQ} -tree \tilde{T} ;

Output: Mapping f from each pair of nonprobes u, v with $\{u, v\} \notin E^+$ to “intersecting”, “potentially intersecting”, or “independent”;

We denote by E_i and E_p the sets of the pairs of intersecting nonprobes, and the pairs of potentially intersecting nonprobes, respectively. That is, each pair of nonprobes u, v is either in E^+ , E_i , E_p , or otherwise, they are independent.

Theorem 6. *The sets E_i and E_p can be computed in $O(|E| + |E^+| + |E_i| + |E_p|)$ time for a given enhanced interval probe graph $G^+ = (P, N, E \cup E^+)$ and the extended \mathcal{MPQ} -tree \tilde{T} .*

Proof. Omitted. □

By Theorem 6, we can heuristically find the “best” nonprobe to fix the structure of the DNA sequence:

Corollary 4. *For a given enhanced interval probe graph $G^+ = (P, N, E \cup E^+)$ and the canonical extended \mathcal{MPQ} -tree \tilde{T} , we can find the nonprobe v that has the most potentially intersecting nonprobes in $O(|E| + |E^+| + |E_i| + |E_p|)$ time.*

We next consider the following problem:

Input: An interval probe graph $G = (P, N, E)$ and the canonical extended \mathcal{MPQ} -tree \tilde{T} ;

Output: All affirmative interval graphs.

Theorem 7. *For a given enhanced interval probe graph $G = (P, N, E)$ and the canonical extended \mathcal{MPQ} -tree \tilde{T} , all affirmative interval graphs can be enumerated in polynomial time and space of $|P| + |N| + |M|$, where M is the number of the affirmative interval graphs.*

Proof. We here show how to generate one possible affirmative interval graph of G . It is easy to modify it to enumerate all affirmative interval graphs in polynomial time and space of $|P| + |N| + |M|$. We first fix each floating leaf as a leaf under the corresponding Q -node (in arbitrary way). Then, we have an affirmative \mathcal{MPQ} -tree for some interval graph. However, to generate all possible interval graphs, we have to consider two more cases; (1) two adjacent nonprobes in N^* might have intersection as noted in Observation 1, and (2) two adjacent nonprobes in N_S might have intersection. Those two cases can be analyzed in the same case-analysis in the proof of Theorem 6. Then we next fix the relations between each pair of nonprobes (We note that some pair of nonprobes u and v may be determined by the relation of the other pair of nonprobes u and w). It is easy to see that for each possible affirmative interval graph, its \mathcal{MPQ} -tree can be generated in this way. □

References

1. C. Berge. *Hypergraphs*. Elsevier, 1989.
2. K.S. Booth and G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ -Tree Algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

3. A. Brandstädt, F.F. Dragan, H.-O. Le, V.B. Le, and R. Uehara. Tree Spanners for Bipartite Graphs and Probe Interval Graphs. In *29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '03)*, pages 106–118. Lecture Notes in Computer Science Vol. 2880, Springer-Verlag, 2003.
4. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
5. C.J. Colbourn and K.S. Booth. Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs. *SIAM Journal on Computing*, 10(1):203–225, 1981.
6. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
7. M.C. Golumbic and M. Lipshteyn. Chordal Probe Graphs. In *29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '03)*, pages 249–260. Lecture Notes in Computer Science Vol. 2880, Springer-Verlag, 2003.
8. M.C. Golumbic and A.N. Trenk. *Tolerance Graphs*. Cambridge studies in advanced mathematics 89. Cambridge, 2004.
9. J.L. Johnson, R.M. McConnell, and J.P. Spinrad. Linear Time Recognition of Probe Interval Graphs. in preparation, 2002.
10. J.L. Johnson and J.P. Spinrad. A Polynomial Time Recognition Algorithm for Probe Interval Graphs. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 477–486. ACM, 2001.
11. N. Korte and R.H. Möhring. An Incremental Linear-Time Algorithm for Recognizing Interval Graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.
12. G.S. Lueker and K.S. Booth. A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.
13. R.M. McConnell and J.P. Spinrad. Construction of Probe Interval Models. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 866–875. ACM, 2002.
14. T.A. McKee and F.R. McMorris. *Topics in Intersection Graph Theory*. SIAM, 1999.
15. F.R. McMorris, C. Wang, and P. Zhang. On Probe Interval Graphs. *Discrete Applied Mathematics*, 88:315–324, 1998.
16. A. Nerry, M.C. Golumbic, and M. Lipshteyn. Two Tricks to Triangulate Chordal Probe Graphs in Polynomial Time. In *Proc. 15th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 955–962. ACM, 2004.
17. J.P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.
18. R. Uehara, S. Toda, and T. Nagoya. Graph Isomorphism Completeness for Chordal bipartite graphs and Strongly Chordal Graphs. *Discrete Applied Mathematics*, 2004. to appear.
19. P. Zhang. Probe Interval Graphs and Its Applications to Physical Mapping of DNA. manuscript, 1994.
20. P. Zhang. Probe Interval Graph and Its Applications to Physical Mapping of DNA. RECOMB 2000, Poster Session; available at <http://recomb2000.ims.u-tokyo.ac.jp/Posters/list-posters.html>, 2000.
21. P. Zhang. United States Patent. Method of Mapping DNA Fragments. [Online] Available <http://www.cc.columbia.edu/cu/cie/techlists/patents/5667970.htm>, July 3 2000.

Efficient Algorithms for the Longest Path Problem

Ryuhei Uehara¹ and Yushi Uno²

¹ Department of Information Processing, School of Information Science,

JAIST, Ishikawa, Japan

uehara@jaist.ac.jp

² Department of Mathematics and Information Science, College of Integrated Arts and Sciences, Osaka Prefecture University, Sakai, Japan

uno@mi.cias.osakafu-u.ac.jp

Abstract. The longest path problem is to find a longest path in a given graph. While the graph classes in which the Hamiltonian path problem can be solved efficiently are widely investigated, very few graph classes are known where the longest path problem can be solved efficiently. For a tree, a simple linear time algorithm for the longest path problem is known. We first generalize the algorithm, and it then solves the longest path problem efficiently for weighted trees, block graphs, ptolemaic graphs, and cacti. We next propose three new graph classes that have natural interval representations, and show that the longest path problem can be solved efficiently on those classes. As a corollary, it is also shown that the problem can be solved efficiently on threshold graphs.

Keywords: Efficient algorithms, graph classes, longest path problem.

1 Introduction

The Hamiltonian path problem (HPP, for short) is one of the most well known \mathcal{NP} -hard problems and has numerous applications [15]. For such an intractable problem, there are two major approaches; approximation algorithms [17, 2, 25] and parameterized complexity [13]. In both approaches, we have to change the decision problem to the optimization problem. Therefore the longest path problem (LPP) is one of the basic problems from the viewpoint of combinatorial optimization. From the practical point of view, it is also a very natural approach to find a longest path in a given graph when it does not have a Hamiltonian path. However, finding a longest path seems to be more difficult than determining whether the given graph has a Hamiltonian path or not. Even if a given graph has a Hamiltonian path, it is impossible to efficiently find a path of length $n - n^\epsilon$ for any $\epsilon < 1$ unless $\mathcal{P} = \mathcal{NP}$ [18]. In general, LPP does not belong to \mathcal{APX} unless $\mathcal{P} = \mathcal{NP}$ [18], and the best known performance ratio of an approximation algorithm is $O(n(\log \log n / \log n)^2)$ [7] (see also [21, 1, 23, 26] for related results).

Until now, many graph classes have been proposed, and the complexity of hard problems over those graph classes have been extensively investigated [9, 16]. The classification of the graph classes by the difficulty in solving HPP gives us an insight for LPP. If HPP is \mathcal{NP} -hard, LPP is also intractable since HPP is a special case of LPP. Such “hard” graph classes include chordal bipartite graphs, strongly chordal split graphs

(and hence chordal graphs and split graphs) [22], undirected path graphs, double interval graphs, rectangle graphs [6], and circle graphs [11]. On the other hand, all proper interval graphs have a Hamiltonian path [5]. Thus we can find a longest path of length equal to the number of vertices for any given proper interval graph. Between them, there are “non-trivial” graph classes; HPP is polynomial time solvable for circular arc graphs (and hence interval graphs) [12], and bipartite permutation graphs [24]. In this paper, we focus on LPP for the “non-trivial” graph classes, and propose efficient algorithms for several graph classes.

There are few polynomial time algorithms for finding a longest path in a graph; as far as the authors know, trees are the only natural and non-trivial graph class in which LPP can be solved in polynomial time. The algorithm for trees was invented by W. Dijkstra around 1960. It runs in linear time, and the formal proof is given by R.W. Bulterman et al [10].

We first generalize Dijkstra’s longest path algorithm (in this paper, we abbreviate this simply as Dijkstra’s algorithm) and its proof in [10], and show that LPP is solved efficiently for (vertex/edge) weighted trees, block graphs, ptolemaic graphs, and cacti.

Next, we focus on graph classes which have natural interval representations. Although all longest paths of a connected interval graph have non-empty intersection [3, Corollary 2.2], there are no efficient algorithms for finding a specific longest path in an interval graph. We introduce three natural and non-trivial graph classes and show that LPP can be solved efficiently on those classes. As a direct corollary, LPP is solved efficiently for threshold graphs.

2 Preliminaries

A graph $G = (V, E)$ consists of a finite set V of *vertices* and a collection E of 2-element subsets of V called *edges*. The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$. For a subset U of V , we denote by $N_G(U)$ the set $\{v \in V \mid v \in N(u) \text{ for some } u \in U\}$. If no confusion can arise we will omit the index G . We denote the closed neighborhood $N(v) \cup \{v\}$ by $N[v]$. Given a graph $G = (V, E)$, its *complement* is defined by $\bar{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$, and is denoted by $\bar{G} = (V, \bar{E})$. A vertex set I is an *independent set* if $G[I]$ contains no edges, and the graph $\bar{G}[I]$ is called a *clique*. A graph $G = (V, E)$ is *bipartite* if V can be partitioned into two independent sets. A graph with a partition of its vertex set V into X and Y can be denoted by $G = (X \cup Y, E)$.

For $G = (V, E)$, a sequence of distinct vertices v_0, v_1, \dots, v_l is a *path*, denoted by (v_0, v_1, \dots, v_l) , if $\{v_j, v_{j+1}\} \in E$ for each $0 \leq j < l$. The *length* of a path is the number of edges on the path. For two vertices u and v , the *distance* of the vertices, denoted by $d(u, v)$, is the minimum length of the paths joining u and v . A *cycle* is a path beginning and ending with the same vertex. A cycle of length i is denoted by C_i . An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of that cycle. A graph is *chordal* if each cycle of length at least 4 has a chord. Given a graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $G[N(v)]$ is a clique in G .

2.1 Graph Classes with Interval Graph Representation

A graph (V, E) with $V = \{v_0, v_1, \dots, v_{n-1}\}$ is an *interval graph* if there is a set of intervals $\mathcal{I} = \{I_{v_0}, I_{v_1}, \dots, I_{v_{n-1}}\}$ such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $0 \leq i, j < n$. We call the set \mathcal{I} of intervals an *interval representation* of the graph. For each interval I , we denote by $L(I)$ and $R(I)$ the left and right endpoints of the interval, respectively (hence we have $L(I) \leq R(I)$ and $I = [L(I), R(I)]$).

A bipartite graph $(X \cup Y, E)$ with $X = \{x_0, x_1, \dots, x_{n_1-1}\}$ and $Y = \{y_0, y_1, \dots, y_{n_2-1}\}$ is an *interval bigraph* if there are families of intervals $\mathcal{I}_X = \{I_{x_0}, I_{x_1}, \dots, I_{x_{n_1-1}}\}$ and $\mathcal{I}_Y = \{I_{y_0}, I_{y_1}, \dots, I_{y_{n_2-1}}\}$ such that $\{x_i, y_j\} \in E$ iff $I_{x_i} \cap I_{y_j} \neq \emptyset$ for each i and j with $0 \leq i < n_1$ and $0 \leq j < n_2$. We also call the families of intervals $(\mathcal{I}_X, \mathcal{I}_Y)$ an *interval representation* of the graph. If no confusion can arise we sometimes unify a vertex v and the corresponding interval I_v , and write $N(I_v)$, $L(v)$. Moreover, if a vertex v corresponds to a point, the vertex v and the corresponding point $L(I_v) = R(I_v)$ are sometimes unified.

For two intervals I and J , we write $I \prec J$ if $L(I) \leq L(J)$ and $R(I) \leq R(J)$. For any interval representation \mathcal{I} and a point p , $N[p]$ denotes the set of intervals that contain the point p .

Given an interval (bi)graph, an interval representation is said to be *compact* if the following conditions hold;

1. for each interval I , $R(I)$ and $L(I)$ are integers, and
2. for each pair of integers p, q with $N[p] \neq \emptyset$ and $N[q] \neq \emptyset$, $N[p] \setminus N[q] \neq \emptyset$ and $N[q] \setminus N[p] \neq \emptyset$.

Lemma 1. *For any given interval graph $G = (V, E)$, there is a linear time algorithm that constructs a compact interval representation \mathcal{I} such that*

1. each simplicial vertex v is a point; that is, $L(I_v) = R(I_v)$,
2. $\bigcup_{I \in \mathcal{I}} I$ is contained in $[0, |V| - 1]$, and
3. each integer point i with $N[i] \neq \emptyset$ corresponds to a distinct maximal clique of G .

Proof. Given an interval graph $G = (V, E)$, we can construct an \mathcal{MPQ} -tree, which is a kind of labeled \mathcal{PQ} -tree, in linear time [19]. From the \mathcal{MPQ} -tree, in a natural way, we can construct a compact interval representation in linear time. This fact can be proved by the induction of the number of nodes and sections in the \mathcal{MPQ} -tree, but it is straightforward and tedious, and is omitted here. We show that the constructed compact interval representation satisfies the conditions.

Let v be any simplicial vertex in G . By the definition, $N(v)$ induces a clique. Thus, by the Helly property (see, e.g., [4]), all vertices u in $N(v)$ share a common point. Therefore, I_v contains this point. If I_v is not an integer point, I_v must contain an interval $[i, i + 1]$ for some integer i . Then, $N[i] = N[i + 1]$, which contradicts the compactness of the interval representation. This yields claim 1.

The maximal cliques of an interval graph G can be linearly ordered so that for each vertex v , the maximal cliques containing v occur consecutively [8]. Since the representation is compact, we have claim 3 immediately. It is well known that chordal graphs, and hence interval graphs, have at most n maximal cliques. This fact with claim 3 implies claim 2. \square

An interval graph is a *proper interval graph* if no two intervals I and J properly contain each other, i.e., if either $I \prec J$ or $J \prec I$ for each I and J .

Let $G = (X \cup Y, E)$ be a bipartite graph. An ordering $<$ of X in G has the *adjacency property* if for each vertex $y \in Y$, $N(y)$ consists of vertices that are consecutive in the ordering $<$ of X . A bipartite graph $G = (X \cup Y, E)$ is *biconvex* if there are orderings of X and Y that fulfill the adjacency property. G is *convex* if there is an ordering of X or Y that fulfills the adjacency property.

Given a biconvex graph $G = (X \cup Y, E)$ with $|X| = n_1$ and $|Y| = n_2$, a compact interval representation $\mathcal{I}(G)$ is constructed as follows: Let $x_0 < x_1 < \dots < x_{n_1-1}$ and $y_0 < y_1 < \dots < y_{n_2-1}$ be the orderings of X and Y that have adjacency property. Let x_i correspond to the integer point i with $0 \leq i < n_1$. For each j with $0 \leq j < n_2$, since each y_j contains consecutive x s, we can make y_j correspond to the interval $[L(y_j), R(y_j)]$ such that $L(y_j) = l$ and $R(y_j) = r$, where x_l and x_r are the minimum and maximum vertices in $N(y_j)$, respectively.

Lemma 2. *For the compact interval representation $\mathcal{I}(G)$ of a given biconvex graph $G = (X \cup Y, E)$, there are two indices j_t and j_b such that (1) $L(y_0) \geq L(y_1) \geq \dots \geq L(y_{j_t})$ and $L(y_{j_t}) \leq L(y_{j_t+1}) \leq \dots \leq L(y_{n_2-1})$, and (2) $R(y_0) \leq R(y_1) \leq \dots \leq R(y_{j_b})$ and $R(y_{j_b}) \geq R(y_{j_b+1}) \geq \dots \geq R(y_{n_2-1})$.*

Proof. We note that $x_0 \in N(y_{j_t})$, $x_{n_1-1} \in N(y_{j_b})$, and we can assume that $j_t \leq j_b$ without loss of generality. If the index j_t does not exist, we have three consecutive vertices, say, $y_1 < y_2 < y_3$ such that $L(y_1) > L(y_2)$ and $L(y_3) > L(y_2)$. Then there is a vertex $x \in X$ such that $y_1, y_3 \in N(x)$ and $y_2 \notin N(x)$. This contradicts the definition of biconvex graphs. Thus j_t exists. The case of index j_b is symmetric. □

A graph $G = (V, E)$ is called *threshold* if there exist nonnegative integers $w(v)$ ($v \in V$) and an integer t such that $\sum_{v \in S} w(v) \leq t$ if and only if S is an independent set of G .

2.2 Tree-Like Graph Classes

We here introduce some graph classes that have similar structure to trees. A graph G is a *block graph* if G is connected and every maximal 2-connected subgraph is a clique. Block graphs can be seen as graphs obtained from trees by replacing each edge by a clique, and the cliques have at most one vertex in common. A graph G is *ptolemaic* if for any vertices u, v, w, x of G , $d(u, v)d(w, x) \leq d(u, w)d(v, x) + d(u, x)d(v, w)$. Ptolemaic graphs can be seen as graphs obtained from trees by replacing each edge by a clique, and the cliques have any number of vertices in common. Thus, a block graph is ptolemaic. A *cactus* is a graph whose block is either an edge or a cycle. Similarly, cacti can be seen as graphs obtained from trees by replacing a part of edges by a cycle, and the cycles have at most one vertex in common. See [9] for further details of these graph classes.

2.3 New Graph Classes

We say that an interval graph $G = (V, E)$ is said to be an *interval biconvex graph* if V can be partitioned into two sets S and Y such that

1. each vertex $x \in S$ is simplicial in G , and
2. the bipartite graph $G' := (S \cup Y, E')$ is a biconvex graph, where $E' := \{\{x, y\} \mid x \in S, y \in Y, \text{ and } \{x, y\} \in E\}$.

Intuitively, interval biconvex graphs G are interval graphs obtained from biconvex graphs G' ; they have common interval representations. We have the following proper inclusion:

Lemma 3. *(Proper interval graphs \cup threshold graphs) \subset interval biconvex graphs \subset interval graphs.*

Proof. The inclusions “proper interval graphs \subset interval biconvex graphs \subset interval graphs” are easy to follow directly from the corresponding definitions. It remains to show that any threshold graph $G = (V, E)$ is an interval biconvex graph. We assume that vertices are ordered with respect to their weights; $w(v_0) \leq w(v_1) \leq \dots \leq w(v_{n-1})$. Let i be the smallest index with $w(v_i) \geq \frac{t}{2}$. We partition V into $V' := \{v_i, \dots, v_{n-1}\}$ and $S := V \setminus V'$. Then we have (1) $G[V']$ is a clique, (2) for each vertex v in S , there is an index $j (> i)$ such that $N(v) = \{v_j, \dots, v_{n-1}\}$, and (3) each vertex in S is simplicial. Thus G is an interval biconvex graph. \square

Lemma 4. *Let $G = (S \cup Y, E)$ be an interval biconvex graph. Let v be a (simplicial) vertex in S . Let $N_S[v]$ be the set $\{v\} \cup (N(v) \cap S)$ in G . Then $N_S[v]$ induces a clique, and for any two vertices v_1 and v_2 in $N_S[v]$, $N_G[v_1] = N_G[v_2]$.*

Proof. Since v is simplicial, $N_S[v]$ induces a clique. Thus we show that $N[v_1] = N[v_2]$ for any v_1 and v_2 in $N_S[v]$. We first note that $v_1 \in N[v_2]$ since v is simplicial. Let w be any vertex in $N[v_1]$. Then, since v_1 is also simplicial, $\{w, v_2\} \in E$, consequently, $w \in N[v_2]$. Hence we have $N[v_1] \subseteq N[v_2]$. Symmetric argument implies $N[v_2] \subseteq N[v_1]$, which completes the proof. \square

Corollary 1. *Let $G = (S \cup Y, E)$ be an interval biconvex graph, and $\mathcal{I}(G)$ be a compact interval representation of G . Then*

- (1) for each vertex v in S , I_v is an integer point, and
- (2) for each vertices u and v in S with $N[u] = N[v]$, I_v and I_u are the same integer point.

We say that a biconvex graph $G = (X \cup Y, E)$ with $Y = \{y_0, \dots, y_{n_2-1}\}$ is proper if $y_0 \prec y_1 \prec \dots \prec y_{n_2-1}$ and is linearly included if $I_{y_0} \subseteq I_{y_1} \subseteq \dots \subseteq I_{y_{n_2-1}}$. Intuitively, a biconvex graph G is proper if $j_0 = j_t$ and $j_b = n_2 - 1$, and G is linearly included if $j_t = n_2 - 1$ in Lemma 2. The motivation for introducing these two graph classes comes from the fact that every biconvex graph can be partitioned into three graphs from these classes. This can be achieved as follows: For any biconvex graph $G = (X \cup Y, E)$ with the two indices j_t and j_b from Lemma 2, we partition Y into $Y_1 := \{y_0, \dots, y_{j_t}\}$, $Y_2 := \{y_{j_t+1}, \dots, y_{j_b-1}\}$, and $Y_3 := \{y_{j_b}, \dots, y_{n_2-1}\}$. Let X_i be the set of vertices in $N(Y_i)$ and let E_i be the set of edges induced by $X_i \cup Y_i$. Then $G_1 := (X_1 \cup Y_1, E_1)$ and $G_3 := (X_3 \cup Y_3, E_3)$ are linearly included biconvex graphs, and $G_2 := (X_2 \cup Y_2, E_2)$ is a proper biconvex graph.

3 Algorithms for Tree-Like Graphs

Given a finite tree T , a longest path can be found in linear time. A simple procedure is invented by E.W. Dijkstra around 1960. The formal correctness proof of the procedure is given in [10]. In order to illustrate our algorithms for tree-like graphs, we first give a framework that allows a generalization of Dijkstra's algorithm by transforming G into another graph G' . For a given graph $G = (V, E)$, suppose that we can construct a new graph $G' = (V', E')$ satisfying the following three conditions:

1. $V \subseteq V'$,
2. for each pair u, v in V , $d_{G'}(u, v)$ equals the length of a longest path joining u and v in G , and
3. for each pair u, v in V , $d_{G'}(u, v)$ is given by the unique shortest path on G' .

Then the following algorithm computes the length of a longest path in G by using the graph G' in $O(|V| + |E| + |V'| + |E'|)$ time and space.

Algorithm TR

Input: Graphs $G = (V, E)$ and $G' = (V', E')$.

Output: The length of a longest path P in G .

- T1. pick any vertex u in V of G' ;
- T2. find a vertex x in V such that $d_{G'}(u, x)$ is largest;
- T3. find a vertex y in V such that $d_{G'}(x, y)$ is largest;
- T4. output the length of a shortest path joining x and y on G' .

For $G' = G$, TR is in fact Dijkstra's algorithm.

Theorem 1. *Algorithm TR computes the length of a longest path in G in linear time.*

Proof. In the proof in [10], the correctness of Dijkstra's algorithm is based on the following three facts; (1) for any vertices u and v , the shortest path joining them gives the longest path joining them, (2) for any vertices u, v and w , we have $d(u, v) \leq d(u, w) + d(w, v)$, and (3) for any vertices u, v and w , we have $d(u, v) = d(u, w) + d(w, v)$ if and only if w is on the path joining u and v .

By condition of G' , for any vertices u and v in V , the length of the shortest path joining u and v on G' is equal to the length of the longest path joining u and v on G . For any vertices u, v , and w in V , we also have $d_{G'}(u, v) \leq d_{G'}(u, w) + d_{G'}(w, v)$. By condition of G' , $d_{G'}(u, v)$ is given by the unique shortest path on G' . Thus, for any vertices u, v and w in V , we have $d_{G'}(u, v) = d_{G'}(u, w) + d_{G'}(w, v)$ if and only if w is on the shortest path joining u and v . Therefore, we can use the same argument in the proof of [10] to show the correctness of Algorithm TR. Since shortest paths in G' are unique, we can use the standard BFS to implement TR in linear time. \square

In the following, we investigate graph classes for which Algorithm TR computes the length of a longest path with given suitable reductions. We note that, in step T4, Algorithm TR outputs the length of a longest path. However, it is easy to modify the specialized algorithms so that they output not just the length but the longest path itself.

Lemma 5. Let $G = (V, E)$ be a tree, and $w : V \cup E \rightarrow \mathbf{Z}_+$ be a weight function of vertices and edges. We define the length of a weighted path (v_1, v_2, \dots, v_k) by $\sum_{i=1}^k w(v_i) + \sum_{i=1}^{k-1} w(\{v_i, v_{i+1}\})$. Then the weighted longest path problem can be solved in $O(|V|)$ time and space.

Proof. We can use Dijkstra’s algorithm straightforwardly. □

Theorem 2. Let $G = (V, E)$ be a block graph. Then the longest path problem can be solved in $O(|V| + |E|)$ time and space.

Proof. Given block graph G , we construct a weighted tree G' as follows: For each maximal clique K of size $k > 2$, we first remove all edges in K , second create a vertex c of weight $k - 3$, and third join c and each vertex in K by an edge of weight 1. We repeat the replacement for all maximal cliques of size > 2 . The resulting graph G' is an integer weighted tree. Let u and v be any two vertices in V . Then the length of a longest path between u and v on G is equal to the length of the weighted path joining u and v on G' . Thus we can use Lemma 5. □

Theorem 3. Let $G = (V, E)$ be a ptolemaic graph. Then the longest path problem can be solved in $O(|V| (|V| + |E|))$ time and space.

Proof. Let K and K' be any two maximal cliques in G with $|K \cap K'| \geq 2$. Then, for any pair of vertices u and v in $K \cup K'$, we can construct a path P of length $|K \cup K'| - 1$ such that its endpoints are u and v and P contains all vertices in $K \cup K'$. We now add edges to $G[K \cup K']$ until $G[K \cup K']$ becomes a clique. It is easy to see that the replacement does not change the length of the longest path of the graph. While G contains two maximal cliques K and K' with $|K \cap K'| \geq 2$, we repeat the above process. Let G' be the resulting graph. Then G' is a block graph and the length of a longest path in G' is equal to the length of a longest path in G . Thus, Theorem 2 completes the proof. □

Theorem 4. Let $G = (V, E)$ be a cactus. Then the longest path problem can be solved in $O(|V|^2)$ time and space.

Proof. We first show a reduction in $O(|V|^3)$ time and space. Let C_k be any cycle of k vertices in G . We replace C_k by a gadget C'_k defined as follows (Fig. 1): For each pair of vertices v_i and v_j on C_k , if $d_{C_k}(v_i, v_j) = h$, C'_k contains a path of length $k - h$ (with auxiliary vertices). We

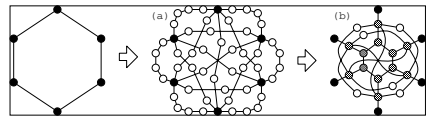


Fig. 1. Gadget for C_5

note that the length of the longest path joining v_i and v_j on G (or C_k) is $k - h$. We replace all cycles of G by the gadgets. The resulting graph G' has $O(|V|^3)$ vertices and edges, and the reduction can be done in $O(|V|^3)$ time and space. To show the correctness of Algorithm TR, we show that for each pair u, v in V , (1) $d_{G'}(u, v)$ equals the length of a longest path joining u and v in G , and (2) the shortest path joining u and v on G' is uniquely determined. Since G is a cactus, we show that for each pair v_i and v_j on C_k , $d_{G'}(v_i, v_j)$ equals the length of a longest path joining v_i and v_j in G . Without

loss of generality, we assume that $i = 0$ and $0 < j \leq \frac{k}{2}$. By the construction, we have $d_{G'}(v_0, v_j)$ is at most $k - j$ by the added path joining v_0 and v_j . We assume that there is a shorter path P in G' joining v_0 and v_j to derive a contradiction. Since G is a cactus, P contains at least one vertex v' on C_k with $v' \neq v_0, v_j$. From the construction, for any pair of vertices u and u' on C'_k , we have $\frac{k}{2} \leq d_{G'}(u, u') \leq k - 1$. Thus, $k - j \leq k - 1$. On the other hand, since P contains three vertices v_0, v_j , and v' on C_k , the length of P is at least $2\frac{k}{2} = k$, which contradicts that the length of P is shorter than $k - j \leq k - 1$. Thus, for each pair u, v in V , $d_{G'}(u, v)$ is equal to the length of a longest path joining u and v in G , and the shortest path joining u and v on G' is uniquely determined. In each gadget in Fig. 1, we replace each path by the edge of weight equal to the length of the path. This reduction can be done in $O(|V|^2)$ time and space. \square

4 Algorithms for Biconvex Graphs

In this section, we consider the longest path problem on two subclasses of biconvex graphs. We assume that a biconvex graph $G = (X \cup Y, E)$ is given with its compact interval representation. For given any path P , we denote by P_X the set of vertices in P and X , and by P_Y the set of vertices in P and Y . We assume that the sets P_X and P_Y are ordered; when $P_X = \{x_{i_0}, x_{i_1}, \dots\}$ and $P_Y = \{y_{j_0}, y_{j_1}, \dots\}$, then $P = (x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \dots)$ or $P = (y_{j_0}, x_{i_0}, y_{j_1}, x_{i_1}, \dots)$.

4.1 An Algorithm for Proper Biconvex Graphs

Lemma 6. *If the graph $G = (X \cup Y, E)$ is proper biconvex, there is a longest path P with $P_X = \{x_{i_0}, x_{i_1}, \dots\}$ and $P_Y = \{y_{j_0}, y_{j_1}, \dots\}$ s.t. $i_k < i_{k+1}$ and $j_k < j_{k+1}$ for each k .*

Proof. Let P' be any longest path $(x_{i'_0}, y_{j'_0}, x_{i'_1}, y_{j'_1}, \dots)$ such that $x_i \in X$ and $y_j \in Y$ (the symmetric case that P' starts from a vertex in Y is omitted). We construct P from P' such that P contains the same vertices in P' and satisfies the condition. The proof is done by the induction for the length of the path. The lemma holds when the length of the path is 2. Thus we assume that the length of the path is at least 3. We have two cases: (1) The path ends two vertices $(x_{i'_k}, y_{j'_k})$ with $x_{i'_k} \in X$ and $y_{j'_k} \in Y$. Then, by the inductive hypothesis, there is a path $P'' = (x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \dots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfying the condition. If either $y_{j_{k-1}} \prec y_{j'_k}$ or $y_{j'_k} \prec y_{j_0}$, the path $(x_{i_0}, y_{j_0}, \dots, x_{i_{k-1}}, y_{j'_{k-1}}, x_{i'_k})$ or $(x_{i'_k}, y_{j'_k}, x_{i_0}, y_{j_0}, \dots, x_{i_{k-1}})$ satisfies the condition. Thus we suppose that $y_{j_0} \prec y_{j'_k} \prec y_{j_{k-1}}$. Then since the graph is proper, there is an index h such that $y_{j_h} \prec y_{j'_k} \prec y_{j_{h+1}}$. Then $x_{i_{h+1}} \in I_{y_{j'_k}}$, and $x_{i'_k} \in (I_{y_{j_h}} \cup I_{y_{j_{h+1}}})$. If $x_{i'_k} \in I_{y_{j_h}}$, the path $(x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \dots, y_{j_h}, x_{i'_k}, y_{j'_k}, x_{i_{h+1}}, y_{j_{h+1}}, \dots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfies the condition. Otherwise, the path $(x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \dots, y_{j_h}, x_{i_{h+1}}, y_{j'_k}, x_{i'_k}, y_{j_{h+1}}, \dots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfies the condition. (2) The path ends two vertices $(y_{j'_{k-1}}, x_{i'_k})$ with $y_{j'_{k-1}} \in Y$ and $x_{i'_k} \in X$. Then, by the inductive hypothesis, there is a path $P'' = (x_{i_0}, y_{j_0}, x_{i_1}, y_{j_1}, \dots, x_{i_{k-1}}, y_{j_{k-1}})$ satisfying the condition. Then we have three possible cases; $y_{j_{k-1}} \prec y_{j'_{k-1}}, y_{j'_{k-1}} \prec y_{j_0}$, and $y_{j_0} \prec y_{j'_{k-1}} \prec y_{j_{k-1}}$. Using the same argument as in (1), we have the lemma. \square

Theorem 5. *In a proper biconvex graph $G = (X \cup Y, E)$, a longest path can be found in $O(|X \cup Y| + |E|)$ time.*

Proof. (Outline.) To compute the longest path satisfying the condition in Lemma 6, we use a standard dynamic programming. We define two functions $f(x_i, y_j)$ and $g(y_j, x_i)$ such that $f(x_i, y_j)$ gives the length of a longest path starting at the edge $\{x_i, y_j\}$ and $g(y_j, x_i)$ gives the length of a longest path starting at the edge $\{y_j, x_i\}$. We define $f(x_i, y_j) = 0$ if $\{x_i, y_j\} \notin E$ or y_j does not exist, and $g(y_j, x_i) = 0$ if $\{x_i, y_j\} \notin E$ or x_i does not exist. Then, if $\{x_i, y_j\} \in E$, we have $f(x_i, y_j) = \max\{f(x_i, y_{j+1}), g(y_j, x_{i+1}) + 1\}$ and $g(y_j, x_i) = \max\{f(x_i, y_{j+1}) + 1, g(y_j, x_{i+1})\}$. Therefore, the length of a longest path is $\max\{f(x_0, y_0), g(y_0, x_0)\}$, which can be computed by a standard dynamic programming in linear time and space. Computing the longest path itself is also straightforward. \square

4.2 An Algorithm for Linearly Included Biconvex Graphs

Lemma 7. *A linearly included biconvex graph $G = (X \cup Y, E)$ is proper.*

Proof. By definition, there is an index i_c such that $N(x_0) \subseteq N(x_1) \subseteq \dots \subseteq N(x_{i_c})$ and $N(x_{n_1-1}) \subseteq N(x_{n_1-2}) \subseteq \dots \subseteq N(x_{i_c})$, where $n_1 = |X|$. Since Y is linearly ordered in inclusion, for any pair of x_i and $x_{i'}$ with $0 \leq i \leq c$ and $c \leq i' \leq n_1 - 1$, we have $N(x_i) \subseteq N(x_{i'})$ or $N(x_{i'}) \subseteq N(x_i)$. Thus, we can linearly order both of X and Y ; the ordering of Y is as it is, and the ordering of X is computed by the following algorithm in linear time:

Input: A linearly included biconvex graph $G = (X \cup Y, E)$ with $|X| = n_1$ and $|Y| = n_2$, the orderings over X and Y , and a compact interval representation $\mathcal{I}(G)$.

Output: A linear ordering of X in inclusion.

P0. $\ell := 0$; $r := n_1 - 1$;

P1. if $N(x_\ell) \subseteq N(x_r)$ then output x_ℓ and $\ell := \ell + 1$ else output x_r and $r := r - 1$;

P2. if $\ell < r$ then goto P1 else output $x_\ell (= x_r)$ and halt.

The correctness of the algorithm is easy. \square

Theorem 6. *In a linearly included biconvex graph $G = (X \cup Y, E)$, a longest path can be found in $O(|X \cup Y| + |E|)$ time.*

Proof. By Lemma 7, G is proper. Since Y is linearly ordered in inclusion, the condition $N(x_\ell) \subseteq N(x_r)$ is equivalent to the condition $\min\{N(x_\ell)\} \geq \min\{N(x_r)\}$. Thus the step P1 in the algorithm in the proof of Lemma 7 can be done in $O(1)$ time, and the algorithm runs in linear time. Therefore, combining the algorithms in Lemma 7 and Theorem 5, we can construct a linear time algorithm that finds a longest path of G . \square

5 An Algorithm for Interval Biconvex Graphs

Let $G = (S \cup Y, E)$ be an interval biconvex graph with the set S of simplicial vertices. Let $x_0 < x_1 < \dots < x_{n_1-1}$ and $y_0 < y_1 < \dots < y_{n_2-1}$ be the orderings over S and Y that have adjacency property. We here denote by $I(x_i)$ the integer point I_{x_i} , that is,

$I(x_i) = R(I_{x_i}) = L(I_{x_i})$. By Lemma 4, for each x and x' with $N[x] = N[x']$, we have $I(x) = I(x')$. For each j with $0 \leq j \leq n_2 - 1$, since each y_j contains consecutive x s, we can let y_j correspond to the interval $[L(y_j), R(y_j)]$ such that $L(y_j) = I(x_\ell)$ and $R(y_j) = I(x_r)$, where x_ℓ and x_r are the minimum and maximum vertices in $N(y_j)$, respectively. By definition and the proof of Lemma 2, we immediately have the following lemma:

Lemma 8. *For the compact interval representation $\mathcal{I}(G)$ of the interval biconvex graph $G = (S \cup Y, E)$ with the set S of simplicial vertices, there are two indices j_t and j_b such that (0) $j_t \leq j_b$, (1) $L(y_0) \geq L(y_1) \geq \dots \geq L(y_{j_t})$ and $L(y_{j_t}) \leq L(y_{j_t+1}) \leq \dots \leq L(y_{n_2-1})$, and (2) $R(y_0) \leq R(y_1) \leq \dots \leq R(y_{j_b})$ and $R(y_{j_b}) \geq R(y_{j_t+1}) \geq \dots \geq R(y_{n_2-1})$.*

The following proposition is also immediate (see also [5, Lemma 2]):

Proposition 1. *Let $G = (S \cup Y, E)$ be a connected interval biconvex graph. Let $y_0 < y_1 < \dots < y_{n_2-1}$ be the orderings over Y that have adjacency property. Then $\{y_{i-1}, y_i\} \in E$ for each $1 \leq i \leq n_2 - 1$. That is, $(y_0, y_1, \dots, y_{n_2-1})$ is a path of G .*

Hereafter, we denote by (S, y) a path (x_1, \dots, x_k, y) for a set $S = \{x_1, \dots, x_k\}$ if $G[S]$ is a clique.

Lemma 9. *Given a connected interval biconvex graph $G = (S \cup Y, E)$, there is a longest path P such that (1) the vertices in $Y \cap P$ appear consecutively; that is, $Y \cap P$ is $y_j, y_{j+1}, \dots, y_{j+k-1}, y_{j+k}$ for some j and k , and those vertices appear according to the ordering over Y , (2) the consecutive vertices in S of P correspond to the same integer point; that is, if P contains a subpath $(y_j, x_i, x_{i+1}, x_{i+2}, \dots, x_{i+h}, y_{j+1})$, we have $I(x_i) = I(x_{i+1}) = \dots = I(x_{i+h})$, (3) the vertices in S of P appear according to the ordering over S ; that is, if $S \cap P$ is $x_{i_1}, x_{i_2}, \dots, x_{i_h}$ in this order, $x_{i_1} < x_{i_2} < \dots < x_{i_h}$, and (4) P starts and ends at the vertices in S .*

Let s and t be integers such that P starts with the vertices in $N[s] \cap S$ and ends with the vertices in $N[t] \cap S$. Let y_s and y_t be the vertices such that P starts with $(N[s], y_s)$ and ends with $(y_t, N[t])$. Then (5) y_s is the minimum vertex in Y with I_{y_s} contains s , and y_t is the maximum vertex in Y with I_{y_t} contains t .

Proof. (1) We assume that P contains y_{j_1}, y_{j_2} , and y_{j_3} in this ordering and $y_{j_1} < y_{j_3} < y_{j_2}$. Then we have five cases; (a) $I(y_{j_1}) \subseteq I(y_{j_3}) \subseteq I(y_{j_2})$, (b) $I(y_{j_1}) \subseteq I(y_{j_3})$ and $y_{j_3} \prec y_{j_2}$, (c) $y_{j_1} \prec y_{j_3} \prec y_{j_2}$, (d) $y_{j_1} \prec y_{j_3}$ and $I(y_{j_2}) \subseteq I(y_{j_3})$, or (e) $I(y_{j_2}) \subseteq I(y_{j_3}) \subseteq I(y_{j_1})$. We assume that they are minimal, that is, they are consecutive in $P \cap Y$. Then, in any case, swapping y_{j_2} and y_{j_3} we have also a path. Repeating this process, we have a path such that the vertices in Y appear according to the ordering over Y . Let y_j and y_{j+k} be the first and last vertices in $P \cap Y$, respectively. We next show that all vertices $y_{j+k'}$ with $0 < k' < k$ appear on P . We assume that some $y_{j+k'}$ with $0 < k' < k$ does not appear on P . Then, inserting it, we have a longer path, which contradicts that P is a longest path.

(2) Let s be any integer such that there is a vertex x in $N[s] \cap S \cap P$. If there is a vertex x' such that $x' \in N[s] \cap S$ and x' does not appear in P , we have longer path by replacing a subpath (x) of P by (x, x') . Thus all vertices in $N[s] \cap S$ appear in P . It is

clear that gathering all vertices in $N[s] \cap S \cap P$ has no effects on the connectivity and length of P . Thus we can assume that all vertices in $N[s] \cap S$ are consecutive in P .

(3) By (1), all vertices in $Y \cap P$ appear consecutively. Thus, using the same argument in (1), we can assume that all vertices in $S \cap P$ also appear consecutively.

(4) To derive a contradiction, we assume that P starts at the vertex y_j in S . Then, by the definition of a compact interval representation, we have (a) $L(y_j) = R(y_{j'})$ for some $j' < j$, or (b) $N[L(y_j)] \cap S \neq \emptyset$. In the case (a), we have longer path by adding $y_{j'}$ at the top of P . On the other hand, in the case (b), we also have longer path by adding the vertices in $N[L(y_j)] \cap S$ at the top of P . Thus, P starts at the vertex in S . Using the same argument, we can show that P ends at the vertex in S .

(5) To derive a contradiction, we assume that P starts with $(N[s], y_s)$ and there is a vertex $y_{s'}$ such that $s' \neq s$, $I_{y_{s'}}$ contains s , and $y_{s'} \prec y_s$. By (1), $y_{s'}$ does not appear in P . However, in the case, we have longer path by replacing $(N[s], y_s)$ by $(N[s], y_{s'}, y_s)$, which is a contradiction. Thus y_s is the minimum vertex in Y with I_{y_s} contains s . Using the symmetric argument, y_t is the maximum vertex in Y with I_{y_t} contains t . \square

By Lemma 9, the outline of our algorithm is as follows:

0. for each integer s and t with $0 \leq s < t < n_1$, suppose $N[s] \cap S$ and $N[t] \cap S$ are the endpoints of a longest path;
1. let y_{j_s} be the smallest vertex with $L[y_{j_s}] \leq s \leq R[y_{j_s}]$, and let y_{j_t} be the largest vertex with $L[y_{j_t}] \leq t \leq R[y_{j_t}]$;
2. for each integer $i = s + 1, s + 2, \dots, t - 1$, determine where the vertices in $S \cap N[i]$ are inserted in the path $(N[s], y_{j_s}, y_{j_{s+1}}, \dots, y_{j_{t-1}}, y_{j_t}, N[t])$.

Step 2 can be solved by finding a maximum weighted matching in the weighted bipartite graph $G' = (X' \cup Y', E')$ defined as follows; $X' = \{i \mid I(x) = i \text{ for some } x \in S \text{ with } s < i < t\}$, $Y' = \{\{y_j, y_{j+1}\} \mid y_j, y_{j+1} \in Y \text{ and } j_s \leq j \leq j_t - 1\}$, and E' consists of edges $e = \{i, \{y_j, y_{j+1}\}\}$ if $N[i]$ contains y_j and y_{j+1} . The weight of the edge $e = \{i, \{y_j, y_{j+1}\}\}$ is defined by $|S \cap N[i]|$. A weighted matching M of G' gives us a path of G as follows; if an edge $e = \{i, \{y_j, y_{j+1}\}\}$ is in M , the path of G contains $(y_j, N[i], y_{j+1})$. By Lemma 9, the maximum weighted matching of G' gives us a longest path of G . The detail of the algorithm can be described as follows:

Algorithm IBG:

Input: An interval biconvex graph $G = (S \cup Y, E)$ with $|S| = n_1$ and $|Y| = n_2$ and a compact interval representation $\mathcal{I}(G)$.

Output: A longest path P .

- B0. construct the weighted bipartite graph $G' = (X', Y', E')$ for G ;
- B1. for each integer s and t with $0 \leq s < t < n_1$ do the following;
 - B1.1. let $N[s]$ and $N[t]$ suppose the endpoints of a longest path;
 - B1.2. let y_{j_s} be the smallest vertex with $L[y_{j_s}] \leq N[s] \leq R[y_{j_s}]$, and let y_{j_t} be the largest vertex with $L[y_{j_t}] \leq N[t] \leq R[y_{j_t}]$;
 - B1.3. let G'' be the induced bipartite graph from G' by the vertices in S between s and t and the vertices in Y between y_{j_s} and y_{j_t} ;
 - B1.4. find a maximum weighted matching M in G'' ;
 - B1.5. compute a path from $(N[s], y_{j_s}, y_{j_{s+1}}, \dots, y_{j_t}, N[t])$ and the weighted edges in M ;
- B2. find the longest path among the paths generated in step B1.5.

Theorem 7. *A longest path in a given interval biconvex graph $G = (S \cup Y, E)$ with $|S \cup Y| = n$ and $|E| = m$ can be found in $O(n^3(m + n \log n))$ time.*

Proof. The correctness of Algorithm IBG follows from Lemma 9. Thus we analyze the running time. For the weighted bipartite graph $G' = (X' \cup Y', E')$ constructed in step B0, we have $|X'| = O(n_1)$, $|Y'| = n_2 - 1$, and $|E'| = O(|E|)$. A maximum weighted matching can be found in $O(|V|(|E| + |V| \log |V|))$ time and $O(|E|)$ space for (any) given graph $G = (V, E)$ [14]. Hence step B1.4 takes $O(n(m + n \log n))$ time with $n = n_1 + n_2$ and $m = |E|$. Therefore total running time is $O(n_1^2 n(m + n \log n)) = O(n^3(m + n \log n))$. \square

Corollary 2. *A longest path in a given threshold graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ can be found in $O(n + m)$ time and space.*

Proof. (Sketch.) By Lemma 3 and Theorem 7, a longest path in a threshold graph can be found in $O(n^3(m + n \log n))$ time. However, Algorithm IBG can be simplified to run in linear time and space using the properties shown in the proof of Lemma 3 and the fact that $G[S]$ is an independent set. A similar idea can be found in [20], and hence is omitted here. \square

6 Concluding Remarks

The major open problem is the complexity of the longest path problem for interval graphs, convex graphs, and biconvex graphs.

References

1. N. Alon, R. Yuster, and U. Zwick. Color-Coding. *J. ACM*, 42(4): 844–856, 1995.
2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, 1999.
3. P.N. Balister, E. Györi, J. Lehel, and R.H. Schelp. Longest Paths in Circular Arc Graphs. Technical report, U. of Memphis, www.msci.memphis.edu/preprint.html, 2002.
4. C. Berge. *Hypergraphs*. Elsevier, 1989.
5. A.A. Bertossi. Finding Hamiltonian Circuits in Proper Interval Graphs. *Info. Proc. Lett.*, 17(2): 97–101, 1983.
6. A.A. Bertossi and M.A. Bonuccelli. Hamiltonian Circuits in Interval Graph Generalizations. *Info. Proc. Lett.*, 23: 195–200, 1986.
7. A. Björklund and T. Husfeldt. Finding a Path of Superlogarithmic Length. *SIAM J. Comput.*, 32(6): 1395–1402, 2003.
8. K.S. Booth and G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *J. Comput. Syst. Sci.*, 13: 335–379, 1976.
9. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
10. R.W. Bulterman, F.W. van der Sommen, G. Zwaan, T. Verhoeff, A.J.M. van Gasteren, and W.H.J. Feijen. On Computing a Longest Path in a Tree. *Info. Proc. Lett.*, 81: 93–96, 2002.
11. P. Damaschke. The Hamiltonian Circuit Problem for Circle Graphs is NP-complete. *Info. Proc. Lett.*, 32: 1–2, 1989.

12. P. Damaschke. Paths in Interval Graphs and Circular Arc Graphs. *Discr. Math.*, 112: 49–64, 1993.
13. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1999.
14. H.N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *Proc. 1st Ann. ACM-SIAM Symp. on Discr. Algo.*, 434–443. ACM, 1990.
15. M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.
16. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs, 2/e*. Ann. Discr. Math., 57. Elsevier, 2004.
17. D. Hochbaum (eds.). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1995.
18. D. Karger, R. Motwani, and G.D.S. Ramkumar. On Approximating the Longest Path in a Graph. *Algorithmica*, 18: 82–98, 1997.
19. N. Korte and R.H. Möhring. An Incremental Linear-Time Algorithm for Recognizing Interval Graphs. *SIAM J. Comput.*, 18(1): 68–81, 1989.
20. N.V.R. Mahadev and U.N. Peled. Longest Cycles in Threshold Graphs. *Discr. Math.*, 135: 169–176, 1994.
21. B. Monien. How to Find Long Paths Efficiently. *Ann. Discr. Math.*, 25: 239–254, 1985.
22. H. Müller. Hamiltonian Circuit in Chordal Bipartite Graphs. *Disc. Math.*, 156: 291–298, 1996.
23. M.G. Scutellà. An Approximation Algorithm for Computing Longest Paths. *European J. Oper. Res.*, 148(3): 584–590, 2003.
24. J. Spinrad, A. Brandstädt, and L. Stewart. Bipartite Permutation Graphs. *Discr. Appl. Math.*, 18: 279–292, 1987.
25. V.V. Vazirani. *Approximation Algorithms*. Springer, 2001.
26. S. Vishwanathan. An Approximation Algorithm for Finding a Long Path in Hamiltonian Graphs. In *Proc. 11th Ann. ACM-SIAM Symp. on Discr. Algo.*, 680–685. ACM, 2000.

Randomized Algorithms for Motif Detection

Lusheng Wang¹, Liang Dong², and Hui Fan³

¹ Department of Computer Science, City University of Hong Kong
Kowloon, Hong Kong

² Department of Computer Science, Peking University, Beijing 100871, P.R. China

³ School of Information and Electronic Engineering,
Institute of Shandong Business and Technology, Yantai, Shandong, P.R. China
lwang@cs.cityu.edu.hk, dongl@theory.cs.pku.edu.cn, fanlinw@263.net

Abstract. Motivation: Motif detection for DNA sequences has many important applications in biological studies, e.g., locating binding sites and regulatory signals, and designing genetic probes etc. In this paper, we propose a randomized algorithm, design an improved EM algorithm and combine them to form a software.

Results: (1) We design a randomized algorithm for consensus pattern problem. We can show that with high probability, our randomized algorithm finds a pattern in polynomial time with cost error at most $\epsilon \times l$ for each string, where l is the length of the motif and ϵ can be any positive number given by the user. (2) We design an improved EM (Expectation Maximization) algorithm that outperforms the original EM algorithm. (3) We develop a software MotifDetector that uses our randomized algorithm to find good seeds and uses the improved EM algorithm to do local search. We compare MotifDetector with Buhler and Tompa's PROJECTION which is considered to be the best known software for motif detection. Simulations show that MotifDetector is slower than PROJECTION when the pattern length is relatively small, and outperforms PROJECTION when the pattern length becomes large.

Availability: Free from <http://www.cs.cityu.edu.hk/~lwang/software/motif/index.html>, subject to copyright restrictions.

1 Introduction

Motif detection for DNA sequences is an important problem in bioinformatics that has many applications in biological studies, e.g., locating binding sites [4], finding conserved regions in unaligned sequences, designing genetic probes [15, 17], etc. The motif detection problem can be defined as follows: given n sequences, each is of length m , and an integer l , where $l \leq m$, find a center string s of length l such that s appears (with some errors) in each of the n given sequences. If no error is allowed, the problem is easy. However, in practice, the occurrence of the center string s in each of the given sequences has mutations and is not exact. The problem becomes extremely hard when errors are allowed. Many mathematic models have been proposed. The following two are important.

The Consensus Pattern Problem: Given n DNA sequences $\{s_1, s_2, \dots, s_n\}$, each is of length m , and an integer l , the \dots asks to find a center string s of length l and a substring t_i of length l in s_i such that

$$\sum_{i=1}^n d(s, t_i)$$

is minimized.

The Closest Substring Problem: Given n DNA sequences $\{s_1, s_2, \dots, s_n\}$, each is of length m , and an integer l , the \dots asks to find a center string s of length l and a substring t_i of length l in s_i such that

$$d = \max_{i=1}^n d(s, t_i)$$

is minimized.

d here is called the \dots . Other measures include the \dots [8] and SP-score.

Other than mathematic models, motif representation is another important issue. There are three representations, \dots , and \dots [7]. Here we focus on consensus patterns and profiles. Let t_1, t_2, \dots, t_n be n strings of length l . Each t_i is an occurrence of a motif. The \dots of the n occurrences is obtained by choosing the letter that appears the most in each of the l columns. The \dots of the n occurrences is a $4 \times l$ matrix W , each cell $W(i, j)$ is a number indicating the occurrence rate of letter i in column j . Figure 1 gives an example.

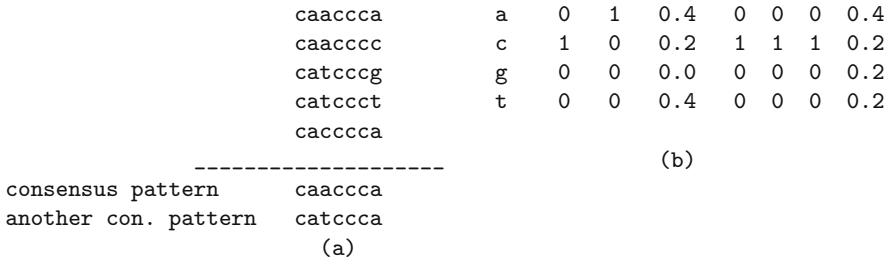


Fig. 1. (a) The 5 occurrences of the motif and the consensus patterns. (b) the profile matrix

To evaluate those mathematic models, representations or programs, Pevzner and Sze [16] proposed a challenge problem, which has been studied by Keich and Pevzner [9, 10]. We randomly generate $n(n = 20)$ sequences of length $m(m = 600)$. Given a center string s of length l , for each of the n random sequences, we randomly choose d positions for s , randomly mutate the d letters from s and implant the mutated copy of s into the random sequence. The problem here is to find the implanted pattern. The pattern thus implanted is called

an (l, d) -... Throughout this paper, we will use this model to do simulations and test our algorithms.

Both the consensus pattern problem and the closest substring problem were proved to be NP-hard and polynomial time approximation schemes have been developed for both problems. However, the closest substring problem seems to be harder than the consensus pattern problem from mathematical point of view. When $l = m$, the consensus pattern problem can be solved in polynomial time, whereas, the closest substring problem becomes the ... problem and still remains NP-hard [11].

In this paper, we adopt the model of consensus pattern. We first design a randomized algorithm for the consensus pattern problem. We show that with high probability, our randomized algorithm finds a pattern in polynomial time with cost error at most $\epsilon \times l$, for any positive number ϵ given by the user. Then, we propose an improved EM algorithm that outperforms the original EM algorithm. Finally, we develop a software MotifDetector that uses our randomized algorithm to find good seeds and uses the improved EM algorithm to do local search. We compare MotifDetector with Buhler and Tompa's PROJECTION [3] That is considered to be the best known software for motif detection. Simulations show that MotifDetector is slower than PROJECTION when l is relatively small, and outperforms PROJECTION when l becomes large.

The improved EM algorithm itself is an independent contribution. It can greatly enhance the speed and quality of the EM algorithm. It can be embedded into other motif detection software.

2 A Randomized Algorithm for Consensus Pattern

For consensus pattern problem, a PTAS was designed in [13, 14]. However, that algorithm is not fast enough to give good solutions in practice. Here we propose a randomized algorithm that works reasonably well in practice.

The technique originates from the PTAS in [14] for the closest substring selection problem. We use it here for the consensus pattern. The idea is to randomly choose k positions among the l positions of the pattern. Then we can guess the true consensus pattern at the k selected positions by trying 4^k possible strings of length k . The partial consensus pattern (with k guessed letters) is used to search all the given sequences s_1, s_2, \dots, s_n to find a t_i from each s_i that is closest to the partial consensus pattern, i.e., the number of mismatches between t_i and the partial consensus pattern at those selected k positions is minimized. Let K be the set of k selected positions, and t'_i and s two strings of length l . We use $d(t'_i, s|K)$ to denote the number of mismatches between t'_i and s at the positions in K . The algorithm is given in Figure 2.

In Step (3), when using a partial consensus string s_p to search a given string s_i , we try to find a substring of length l such that the number of mismatches at those k selected positions is minimized. Steps (1)-(4) generate a candidate of the center string. Since this is a randomized algorithm, different executions generate

1. randomly choose k positions from the l positions for the center string;
2. try all 4^k possible (partial) consensus strings;
3. use the partial consensus strings s_p to search all the n given strings and find a substring that is closest to the partial center string from each of the n given strings;
4. reconstruct the center string s based on the n selected substrings of length l ;
5. repeat 1–4 several times and choose the best result.

Fig. 2. A randomized algorithm for consensus pattern

different results. Thus, in the algorithm, we repeat Steps (1)-(4) several times to enhance the quality of the output.

An interesting problem is how big the parameter k should be. k is related to the accuracy of the solution. Let t_i be the true occurrence of the motif in the given string s_i and t'_i be a substring of length l in s_i . s denotes the consensus pattern. Assume that t'_i is quite different from t_i , say,

$$d(s, t'_i) \geq d(s, t_i) + 2\epsilon l \tag{1}$$

for a small number ϵ that represents the error rate. $Pr(d(t'_i, s|K) \leq d(s, t_i|K))$ denotes the probability that $d(t'_i, s|K) \leq d(s, t_i|K)$. We will estimate $Pr(d(t'_i, s|K) \leq d(s, t_i|K))$, the probability that t'_i is chosen to be the closest substring to the partial consensus pattern $s|K$. Note that from (1), $d(t'_i, s|K) \leq d(s, t_i|K)$ implies either $d(s, t'_i|K) \leq (d(s, t'_i) - \epsilon l) \times \frac{k}{l}$ or $d(s, t_i|K) \geq (d(s, t_i) + \epsilon l) \times \frac{k}{l}$. Thus, we have

$$Pr(d(t'_i, s|K) \leq d(s, t_i|K)) \leq Pr(d(t'_i, s|K) \leq (d(s, t'_i) - \epsilon l) \times \frac{k}{l}) + Pr(d(s, t_i|K) \geq (d(s, t_i) + \epsilon l) \times \frac{k}{l}). \tag{2}$$

Note that $d(t'_i, s|K)$ is the sum of k independent random 0-1 variables $\sum_{i=1}^k X_i$, where $X_i = 1$ indicating a mismatch between s and t'_i at the i -th position in K . Thus, from Chernoff's bounds,

$$Pr(d(t'_i, s|K) \leq (d(s, t'_i) - \epsilon l) \times \frac{k}{l}) \leq \exp(-\frac{1}{2}\epsilon^2 k).$$

If $k = \lceil \frac{4}{\epsilon^2} \log(nm) \rceil$, then

$$Pr(d(t'_i, s|K) \leq (d(s, t'_i) - \epsilon l) \times \frac{k}{l}) \leq (nm)^{-2}.$$

Similarly, it can be proved that if $k = \lceil \frac{4}{\epsilon^2} \log(nm) \rceil$,

$$Pr(d(s, t_i|K) \geq (d(s, t_i) + \epsilon l) \times \frac{k}{l}) \leq (nm)^{-\frac{4}{3}}.$$

Therefore,

$$Pr(d(t'_i, s|K) \leq d(s, t_i|K)) \leq 2(nm)^{-\frac{4}{3}}.$$

Consider all the $n(m - l + 1)$ substrings of length l in the n given strings, we know that with probability $1 - 2(nm)^{-\frac{1}{3}}$, $\sum_{i=1}^n d(s, t'_i) \leq \sum_{i=1}^n (d(s, t_i) + 2\epsilon l)$. Therefore we have the following theorem.

Theorem 1. $k = \lceil \frac{4}{\epsilon^2} \log(nm) \rceil$, $1 - 2(nm)^{-\frac{1}{3}}$, $\sum_{i=1}^n d(s, t'_i) \leq \sum_{i=1}^n (d(s, t_i) + 2\epsilon l)$. $O(4^k nml)$

Theorem 1 shows that when k is large, we can get a good approximation solution. However, in practice, k has to be a relatively big number in order to get satisfactory results. The speed is far below that of PROJECTION. PROJECTION uses a random projection method to find seeds and uses an EM method to do local search. In the next section, we propose a combined approach that uses the EM algorithm to replace Steps (3) and (4) in Figure 2.

3 An Randomized Algorithm Using EM Method

Our method contains two parts. (1) we choose a set of starting points, each a $4 \times l$ weight matrix W , representing the initial guess of the motif (using a randomized algorithm). (2) we use the “EM” method to refine the motif. Repeat the above two steps several times and report the best motif found.

3.1 Choosing Starting Points

A set of starting points are generated using the following randomized algorithm:

1. Choose k different positions uniformly at random from l positions.
2. For each of the 4^k possible strings of length k , a matrix W is formed as follows: for column x , if position x is among the k selected positions, then set $W(b, x) = 1$, where b is the letter at position x , and set the other 3 elements in column x to be 0; otherwise, set $W(b, x) = 0.25$ for $b = A, C, G, T$.

Here we use the profile representation of the motif.

3.2 Refining Starting Points with EM Method

Lawrence and Reilly [12] were the first to introduce the Expectation Maximization (EM) algorithm in motif finding problems. Bailey and Elkan [1] used it in multiple motif finding. Buhler and Tompa [3] adopted the EM method in their PROJECTION algorithm in the motif refining step. The following description of the EM algorithm is based on [1].

Let a $4 \times l$ matrix W be the initial guess of the motif. $s_i(j)$ denotes the j -th letter in sequence s_i . Here is the standard EM algorithm to refine the motif:

1. For each position j in each sequence s_i , $s_{ij} = s_i(j)s_i(j+1) \dots s_i(j+l-1)$ denotes the l -mer (substring of length l) starting at $s_i(j)$ and ending at $s_i(j+l-1)$. Calculate the likelihood that s_{ij} is the occurrence of the motif as follows:

$$P(i, j) = \prod_{x=1}^l W(s_{ij}(x), x), \quad 1 \leq i \leq n, \quad 1 \leq j \leq m-l+1$$

where $s_{ij}(x) = s_i(j+x-1)$ is the x -th base in l -mer s_{ij} . In order to avoid zero weights, a fixed small number (we use 0.1) is added to every element of W before calculating the likelihoods.

2. For each l -mer s_{ij} , we get a normalized probability from the likelihood.

$$P'(i, j) = \frac{P(i, j)}{\sum_{j=1}^{m-l+1} P(i, j)}.$$

Replace $P(i, j)$ with $P'(i, j)$.

(The normalization guarantees that $\sum_{j=1}^{m-l+1} P'(i, j) = 1$, reflecting the fact that there is exactly one motif occurrence in each sequence.)

3. Re-estimate the (motif) matrix W from all the l -mers as follows:

$$W = \sum_{i=1}^n \sum_{j=1}^{m-l+1} W^{ij},$$

where W^{ij} is also a $4 \times l$ matrix, constructed from s_{ij} :

$$W^{ij}(b, x) = \begin{cases} P'(i, j) & : \text{ if } b = s_{ij}(x) \\ 0 & : \text{ otherwise.} \end{cases}$$

4. A normalization is applied to W to ensure that the sum of each column in W is 1, i.e.,

$$W'(b, x) = \frac{W(b, x)}{\sum_{b=A,C,G,T} W(b, x)}.$$

Replace W with W' .

5. Steps 1–4 is called a cycle. Let W_{q-1} and W_q be the two consecutive matrices produced in cycles $q-1$ and q . If

$$\max |W_q(b, x) - W_{q-1}(b, x)| < \epsilon, \tag{3}$$

then EM stops. Otherwise, goto step 1 and start next cycle.

In step 5, ϵ is a parameter given by the user. We use a relatively large value $\epsilon = 0.05$ such that on average the EM algorithm stops within very few cycles.

3.3 Reporting the Best Motif

For each starting point, at the termination of the EM algorithm, we get $n(m - l + 1)$ probabilities $P(i, j)$ standing for the likelihood of l -mer s_{ij} being the motif occurrence. We always choose the s_{ij} with the highest $P(i, j)$ as the occurrence of the motif in sequence s_i . Buhler and Tompa [3] use the product of the probabilities:

$$Pro = \prod_{i=1}^n \max_j P(i, j)$$

to measure the quality of the t occurrences of the motif in the n given sequences. We call Pro the *product of probabilities*. It works well for the easy motifs like (11,2), (13,3) and (15,4). However, for some hard motifs such as (19,6) and (14,4), if ϵ in (3) is set to be large (0.05), the correct motif often has a lower probability score than some incorrect ones when the EM algorithm stops.

Though setting a smaller value to ϵ allows the EM algorithm to find the correct motif eventually, the running time of the whole algorithm will be significantly increased. So another choice is to use the measure

$$d = \max_i \min_j d(s_{ij}, c)$$

where $d(\cdot)$ is the hamming distance and c is the consensus string obtained from W by choosing the letter with the biggest probability in each of the l columns. Here for each s_i , we select s_{ij} that is the closest to c and we hope that for each s_i , $\min_j d(s_{ij}, c) \leq d$ for some given number d . An (l, d) motif should have a mismatch-number at most d .

3.4 The Whole Algorithm

Now we can describe the whole algorithm. See Figure 3.

1. for $trial = 1$ to $maxtrials$ do
2. choose k positions from l positions uniformly at random;
3. generate 4^k starting points;
4. for $i = 1$ to 4^k do
5. refine the i -th starting point;
6. if an (l, d) motif is found then goto 7;
7. report the best motif ever found.

Fig. 3. The whole algorithm

The algorithm stops as soon as it finds an (l, d) motif. If such a motif can not be found, it stops after $maxtrials$ iterations. The parameter k is usually set to be 4.

4 Improved EM Algorithm

In this section, we propose some techniques to improve the EM algorithm.

4.1 Threshold

The algorithm proposed in Section 3 spends most of the time on running the EM algorithm. Thus, accelerating the EM algorithm is important. In the construction of W , those matrices $W^{i,j}$ with very small $P(i, j)$ values represent noise and should be eliminated. Therefore, we use the average value of all $P(i, j)$ for $j = 1, 2, \dots, m - l + 1$, i.e., $\frac{1}{m-l+1}$, as the threshold in Step 3 of the algorithm in Section 3.2. An l -mer s_{ij} takes part in the re-estimation of the motif matrix W only when its probability $P(i, j) \geq \frac{1}{m-l+1}$. Thus, Step 3 in Section 3.2 becomes

$$W = \sum_{i=1}^n \sum_{j \in G} W^{ij},$$

where $G = \{j \mid 1 \leq j \leq m - l + 1, P(i, j) \geq \frac{1}{m-l+1}\}$.

After introducing the threshold, we observe the following improvements:

(1) each EM cycle takes less time than before. Table 1 shows the average running time of every EM cycle for different length of motifs, on a 500 MHz Sun UltraSPARC-IIe workstation. (All the simulations in this paper are done on this computer.)

Table 1. Average cycle time (milliseconds). It increases when l increases. The average time for a cycle decreases by about 30%

l	11	13	15	17	19
without threshold	22.4	25.5	28.6	31.5	35.2
with threshold	16.1	18.2	20.0	22.0	23.8

(2) EM converges faster. In our experiments, the average number of cycles for the EM algorithm to stop is reduced from 6.0 to 3.6 after using the threshold, which means a time saving of 40%.

(3) the accuracy of the algorithm, i.e., the probability that the EM algorithm finds the correct motif, also increases. Table 2 illustrates the improvement of the accuracy. (Other tables are omitted due to space limit.) In each case, the program runs on 100 random instances. The accuracy and the average running time for different *maxtrials* is reported. Here 20 sequences, each of length 600, are used.

Table 2. (17,5)-motif

	<i>maxtrials</i>	10	20	30
accuracy (%)	without threshold	28%	48%	60%
	with threshold	59%	82%	89%
average time (seconds)	without threshold	423	720	921
	with threshold	119	161	195

From these tables we can see that adding the threshold can improve both the accuracy and the speed of the algorithm. The improvement on speed is more significant when accuracy requirement increases since the EM algorithm with threshold can find the correct motif earlier and thus stops earlier.

4.2 Shifting

We have often observed the following phenomenon in experiments:

```
tgtgggtcacc  is the actual motif consensus;
ctgtgggtcac  is the motif consensus found by the algorithm.
```

The algorithm spend some time on finding good, but not the correct motif. Those detected patterns have long overlaps with the correct motif. Thus, we can shift the detected motifs with good scores to the left and right for a few positions and see if there is any improvement.

Suppose a motif and its n occurrences are found. Given a number h for shifting, a new motif is produced in this way:

1. For each occurrence s_{ij} of the motif, replace it with $s_{i(j+h)}$.
2. Construct the matrix W based on the new occurrences, and refine it with the EM algorithm.

In our experiment, h is set to be ± 1 and ± 2 . The original motif is treated as $h = 0$.

The algorithm is given in Figure 4.

```
1. for trial = 1 to maxtrials do
2.   choose k positions from l positions uniformly at random;
3.   generate  $4^k$  starting points;
4.   for i = 1 to  $4^k$  do
5.     refine the i-th starting point;
6.     if an (l, d) motif is found then goto 9;
7.   shift the top 10 results in this trial;
8.   if an (l, d) motif is found then goto 9;
9. report the best motif ever found.
```

Fig. 4. The whole algorithm with shifting

Table 3 illustrates the improvement using the shifting technique. (Other tables are omitted due to space limit.)

Table 3. (15,4)-motif

	<i>maxtrials</i>	2	4	8
accuracy (%)	without shift	27%	43%	68%
	with shift	70%	94%	100%
average time (seconds)	without shift	29	50	78
	with shift	27	34	36

We can see that the shifting technique can improve both the accuracy and the speed of the algorithm. The improvement on speed is more significant on a large *maxtrials*. On a small *maxtrials* the shifting technique sometimes even increases the running time a little bit.

5 Implementation of the Software

We put all the techniques together and produce a software, MotifDetector. The program is written in Java 1.1. MotifDetector allows the users to input their own data. The sequences should be in FASTA format. In other words, each sequence has a title line starting with “>” followed by one or more characters (as the title of the sequence) and terminated with an end-of-line. The following lines are assumed to be the sequence until an end-of-file, a blank line, or another line beginning with “>” is encountered. The users can either directly type the sequences in the input area or copy the data from a file and paste them to the input area.

6 Comparison with PROJECTION

In [3], Buhler and Tompa developed a software, PROJECTION, using a random projection algorithm. To our knowledge, this is the best known software for motif detection. PROJECTION can solve the (15,4)-motif challenge problem efficiently, and it can even solve (14,4)-motif, given plenty of time. Figure 5 is an outline of the random projection algorithm.

1. for $trial = 1$ to $maxtrials$ do
2. choose k positions from l positions uniformly at random;
3. hash all the l -mers of the input sequences into 4^k buckets;
4. pick up those buckets that receive at least s l -mers;
5. refine those buckets using EM;
6. if an (l, d) motif is found then goto 7;
7. report the best motif ever found.

Fig. 5. The random projection algorithm

Simulations on various cases have been done to compare our software with PROJECTION. The parameters for PROJECTION are set as in [3] whenever possible. PROJECTION’s `... ..` is enabled, which improved its performance noticeably. See Table 4 for the result.

From Table 4 we can see: (1) PROJECTION runs faster on (11,2), (13,3), (15,4), (10,2), (12,3), (9,1) and (8,1) motifs; (2) our program is both faster and more accurate on the other 6 cases. It seems that PROJECTION is more efficient on short motifs, and our program is good at solving long and subtle motifs.

Table 4. Performance comparison with PROJECTION. On each motif problem, both programs take the same 100 random instances as input. Each problem instance consists of 20 sequences each of length 600. For PROJECTION, set $k = 7$ and $s = 4$. For our program, set $k = 4$, except for (18,6)-motif, on which we found $k = 5$ is better. Running time (in seconds) is averaged on all the 100 instances

motif	PROJECTION			our program			
	m	accuracy	time	k	m	accuracy	time
(11, 2)	6	100%	5.6	4	3	100%	13
(13, 3)	12	100%	12	4	4	100%	21
(15, 4)	30	100%	26	4	8	100%	36
(10, 2)	60	100%	25	4	30	100%	47
(12, 3)	300	99%	203	4	70	99%	207
(17, 5)	160	99%	83	4	16	100%	52
(19, 6)	200	99%	194	4	30	100%	92
(14, 4)	800	89%	936	4	160	90%	813
(16, 5)	1200	76%	2334	4	300	82%	2063
(18, 6)	2400	81%	4929	5	150	89%	4661
(9, 1)	1	100%	4.3	4	1	100%	6.9
(8, 1)	3	100%	4.3	4	5	100%	9.6
(21, 7)	400	96%	332	4	60	100%	102

7 Conclusions

We have developed a software MotifDetector for motif detection. Simulations show that MotifDetector is slower than PROJECTION when the pattern length is relatively small, and outperforms PROJECTION when the pattern length becomes large.

Another contribution is an improved EM algorithm. It can be applied in many current algorithms such as MEME and PROJECTION. We can expect that after using the improved EM algorithm, their performances might be significantly improved.

Acknowledgement

The work is fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1047/01E].

References

1. Bailey, T. and Elkan, C. (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization, *Machine Learning*, 21, pp.51–80.

2. Blanchette, M. (2001) Algorithms for phylogenetic footprinting. *RECOMB 01: Proceedings of the Fifth Annual International Conference on Computational Molecular Biology*, pp.49–58.
3. Buhler, J. and Tompa, M. (2002) Finding motifs using random projections. *Journal of Computational Biology*, 9, pp.225–242.
4. Dopazo, J., Rodríguez, A., Sáiz, J.C. and Sobrino, F. (1993) Design of primers for PCR amplification of highly variable genomes, *CABIOS*, 9, pp.123–125.
5. Duret, L. and Bucher, P. (1997) Searching for regulatory elements in human non-coding sequences. *Curr. Opin. Struct. Biol.*, 7, pp.399–406.
6. Duret, L., Dorkeld, F. and Gautier, C. (1993) Strong conservation of non-coding sequences during vertebrates evolution: potential involvement in post-transcriptional regulation of gene expression. *Nucleic Acids Research*, 21, pp.2315–2322.
7. Gusfield, D. (1997) Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, *Cambridge University Press*.
8. Hertz, G. and Stormo, G. (1995) Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps, *Proc. 3rd Int'l Conf. Bioinformatics and Genome Research*, pp.201–216.
9. Keich, U. and Pevzner, P. (2002a) Finding motifs in the twilight zone. *Bioinformatics*, 18, pp.1374–1381.
10. Keich, U. and Pevzner, P. (2002b) Subtle motifs: defining the limits of motif finding algorithms. *Bioinformatics*, 18, pp.1382–1390.
11. Lanctot, K., Li, M., Ma, B., Wang, S. and Zhang, L. (1999) Distinguishing string selection problems, *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pp.633–642. Also to appear in *Information and Computation*.
12. Lawrence, C. and Reilly, A. (1990) An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences, *Proteins*, 7, pp.41–51.
13. Li, M., Ma, B. and Wang, L. (2002) Finding Similar Regions in Many Sequences, *J. Comput. Syst. Sci.*, 65, pp.73–96, special issue for *Thirty-first Annual ACM Symposium on Theory of Computing*.
14. Li, M., Ma, B. and Wang, L. (2002) On the closest string and substring problems, *JACM*, 49(2): pp.157–171.
15. Lucas, K., Busch, M., Mössinger, S. and Thompson, J.A. (1991) An improved micro-computer program for finding gene- or gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes, *CABIOS*, 7, pp.525–529.
16. Pevzner, P. and Sze, S. (2000) Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*. pp.269–278.
17. Proutski, V. and Holme, E.C. (1996) Primer Master: a new program for the design and analysis of PCR primers, *CABIOS*, 12, pp.253–255.

Weighted Coloring on Planar, Bipartite and Split Graphs: Complexity and Improved Approximation

Dominique de Werra¹, Mare Demange², Bruno Escoffier³,
Jerome Monnot³, and Vangelis Th. Paschos³

¹ Ecole Polytechnique Fédérale de Lausanne, Switzerland

dewerra@ima.epfl.ch

² ESSEC, Dept. SID, France

demange@essec.fr

³ Université Paris Dauphine, LAMSADE, CNRS UMR 7024, 75016 Paris, France
{escoffier, monnot, paschos}@lamsade.dauphine.fr

Abstract. We study complexity and approximation of MIN WEIGHTED NODE COLORING in planar, bipartite and split graphs. We show that this problem is **NP**-complete in planar graphs, even if they are triangle-free and their maximum degree is bounded above by 4. Then, we prove that MIN WEIGHTED NODE COLORING is **NP**-complete in P_8 -free bipartite graphs, but polynomial for P_5 -free bipartite graphs. We next focus ourselves on approximability in general bipartite graphs and improve earlier approximation results by giving approximation ratios matching inapproximability bounds. We next deal with MIN WEIGHTED EDGE COLORING in bipartite graphs. We show that this problem remains strongly **NP**-complete, even in the case where the input-graph is both cubic and planar. Furthermore, we provide an inapproximability bound of $7/6 - \varepsilon$, for any $\varepsilon > 0$ and we give an approximation algorithm with the same ratio. Finally, we show that MIN WEIGHTED NODE COLORING in split graphs can be solved by a polynomial time approximation scheme.

1 Introduction

We give in this paper some complexity results as well as some improved approximation results for MIN WEIGHTED NODE COLORING, originally studied in Guan and Zhu [7] and more recently in [4]. A k -coloring of $G = (V, E)$ is a partition $\mathcal{S} = (S_1, \dots, S_k)$ of the node set V of G into stable sets S_i . In this case, the objective is to determine a node coloring minimizing k . A natural generalization of this problem is obtained by assigning a strictly positive integer weight $w(v)$ for any node $v \in V$, and defining the weight of stable set S of G as $w(S) = \max\{w(v) : v \in S\}$. Then, the objective is to determine $\mathcal{S} = (S_1, \dots, S_k)$ a node coloring of G minimizing the quantity $\sum_{i=1}^k w(S_i)$. This problem is easily shown **NP**-hard; it suffices to consider $w(v) = 1, \forall v \in V$ and MIN WEIGHTED NODE COLORING becomes the classical node coloring problem. Other versions of weighted colorings have been studied in Hassin and Monnot [8].

Consider an instance I of an **NP**-hard optimization problem Π and a polynomial time algorithm A computing feasible solutions for Π . Denote by $m_A(I, S)$ the value of a Π -solution S computed by A on I and by $\text{opt}(I)$, the value of an optimal Π -solution for I . The quality of A is expressed by the ratio (called approximation ratio in what follows) $\rho_A(I) = m_A(I, S)/\text{opt}(I)$, and the quantity $\rho_A = \inf\{r : \rho_A(I) < r, I \text{ instance of } \Pi\}$. A very favourable situation for polynomial approximation occurs when an algorithm achieves ratios bounded above by $1 + \varepsilon$, for any $\varepsilon > 0$. We call such algorithms *polynomial time approximation schemes*. The complexity of such schemes may be polynomial or exponential in $1/\varepsilon$ (they are always polynomial in the sizes of the instances). A polynomial time approximation scheme with complexity polynomial also in $1/\varepsilon$ is called *polynomial time approximation scheme*.

This paper extends results on **MIN WEIGHTED NODE COLORING**, the study of which has started in Demange et al. [4]. We first deal with planar graphs and we show that, for this family, the problem studied is **NP**-complete, even if we restrict to triangle-free planar graphs with node-degree not exceeding 4.

We then deal with particular families of bipartite graphs. The **NP**-completeness of **MIN WEIGHTED NODE COLORING** has been established in [4] for general bipartite graphs. We show here that this remains true even if we restrict to planar bipartite graphs or to P_{21} -free bipartite graphs (for definitions graph-theoretical notions used in this paper, the interested reader is referred to Berge [1]). It is interesting to observe that these results are obtained as corollaries of a kind of generic reduction from the precoloring extension problem shown to be **NP**-complete in Bodlaender et al. [2], Hujter and Tuza [10, 11], Kratochvil [13]. Then, we slightly improve the last result to P_8 -free bipartite graphs and show that the problem becomes polynomial in P_5 -free bipartite graphs. Observe that in [4], we have proved that **MIN WEIGHTED NODE COLORING** is polynomial for P_4 -free graphs and **NP**-complete for P_5 -free graphs.

Then, we focus ourselves on approximability of **MIN WEIGHTED NODE COLORING** in (general) bipartite graphs. As proved in [4], this problem is approximable in such graphs within approximation ratio $4/3$; in the same paper a lower bound of $8/7 - \varepsilon$, for any $\varepsilon > 0$, was also provided. Here we improve the approximation ratio of [4] by matching the $8/7$ -lower bound of [4] with a same upper bound; in other words, we show here that **MIN WEIGHTED NODE COLORING** in bipartite graphs is approximable within approximation ratio bounded above by $8/7$.

We next deal with **MIN WEIGHTED EDGE COLORING** in bipartite graphs. In this problem we consider an edge-weighted graph G and try to determine a partition of the edges of G into matchings in such a way that the sum of the weights of these matchings is minimum (analogously to the node-model, the weight of a matching is the maximum of the weights of its edges). In [4], it is shown that **MIN WEIGHTED EDGE COLORING** is **NP**-complete for cubic bipartite graphs. Here, we slightly strengthen this result showing that this problem remains strongly **NP**-complete, even in cubic and planar bipartite graphs. Furthermore, we strengthen the inapproximability bound provided in [4], by reducing it from $8/7 - \varepsilon$ to

$7/6 - \varepsilon$, for any $\varepsilon > 0$. Also, we match it with an upper bound of the same value, improving so the $5/3$ -approximation ratio provided in [4].

Finally, we deal with approximation of MIN WEIGHTED NODE COLORING in split graphs. As proved in [4], MIN WEIGHTED NODE COLORING is strongly NP-complete in such graphs, even if the nodes of the input graph receive only one of two distinct weights. It followed that this problem cannot be solved by fully polynomial time approximation schemes, but no approximation study was addressed there. In this paper we show that MIN WEIGHTED NODE COLORING in split graphs can be solved by a polynomial time approximation scheme.

In the remainder of the paper we shall assume for any weighted node or edge coloring $\mathcal{S} = (S_1, \dots, S_\ell)$ considered, we will have $w(S_1) \geq \dots \geq w(S_\ell)$.

2 Weighted Node Coloring in Triangle-Free Planar Graphs

The node coloring problem in planar graphs has been shown NP-complete by Garey and Johnson [5], even if the maximum degree does not exceed 4. On the other hand, this problem becomes easy in triangle-free planar graphs, (see Grotzsch [6]). Here, we show that the weighted node coloring problem is NP-complete in triangle-free planar graphs with maximum degree 4 by using a reduction from 3-SAT PLANAR, proved to be NP-complete in Lichtenstein [14]. This problem is defined as follows: Given a collection $\mathcal{C} = (C_1, \dots, C_m)$ of clauses over the set $X = \{x_1, \dots, x_n\}$ of Boolean variables such that each clause C_j has at most three literals (and at least two), is there a truth assignment f satisfying \mathcal{C} ? Moreover, the bipartite graph $BP = (L, R; E)$ is planar where $|L| = n$, $|R| = m$ and $[x_i, c_j] \in E$ iff the variable x_i (or \bar{x}_i) appears in the clause C_j .

Theorem 1. MIN WEIGHTED NODE COLORING .. NP

Let $BP = (L, R; E)$ be the bipartite graph representing an instance (X, \mathcal{C}) of 3-SAT PLANAR where $L = \{x_1, \dots, x_n\}$, $R = \{c_1, \dots, c_m\}$. We construct an instance $I = (G, w)$ of MIN WEIGHTED NODE COLORING by using two gadgets: The gadgets clause $F(C_j)$ are given in Figure 1 for clause C_j of size 3 and in Figure 2 for clause C_j of size 2. The nodes c_j^k are those that will be linked to the rest of the graph.

The gadgets variable $H(x_i)$ is given in Figure 3 for variable x_i . Assume that x_i appears p_1 times positively and p_2 times negatively in (X, \mathcal{C}) , then in $H(x_i)$ there are $2p = 2(p_1 + p_2)$ special nodes $x_i^k, \bar{x}_i^k, k = 1, \dots, p$. These nodes form a path which meets nodes x_i^k, \bar{x}_i^k alternately.

The weight of nodes which are not given in Figures 1, 2 and 3 are 1. These gadgets are linked together by the following process. If variable x_i appears positively (resp. negatively) in clause c_j , we link one of the variables \bar{x}_i^k (resp. x_i^k), with a different k for each C_j , to one of the three nodes c_j^l of gadget $F(C_j)$. This can be done in a way which preserves the planarity of the graph. Observe that G is triangle-free and planar with maximum degree 4. Moreover, we assume

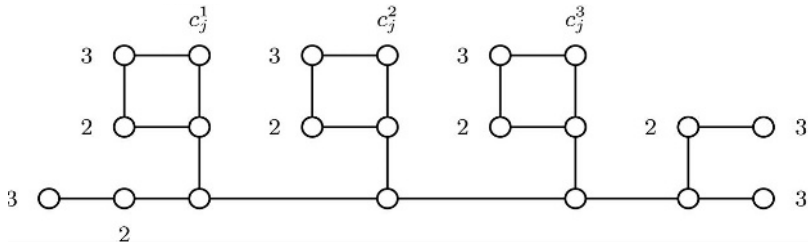


Fig. 1. Graph $F(C_j)$ representing a clause C_j of size 3

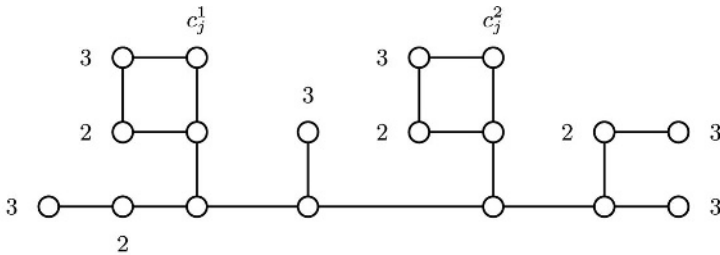


Fig. 2. Graph $F(C_j)$ representing a clause C_j of size 2

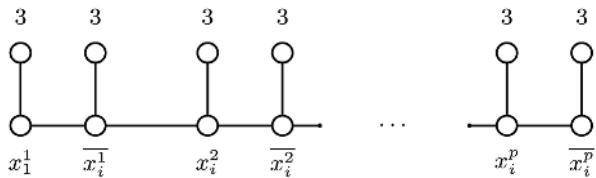


Fig. 3. Graph $H(x_i)$ representing variable x_i

that G is not bipartite (otherwise, we add a disjoint cycle Γ with $|\Gamma| = 7$ and $\forall v \in V(\Gamma), w(v) = 1$).

It is then not difficult to check that (X, \mathcal{C}) is satisfiable iff $opt(I) \leq 6$.

3 Weighted Node Coloring in Bipartite Graphs

3.1 Complexity Results

The NP-completeness of MIN WEIGHTED NODE COLORING in bipartite graphs has been proved in [4]. Here, we show that some more restrictive versions are also NP-complete, namely bipartite planar graphs and P_8 -free bipartite graphs, i.e. bipartite graphs which do not contain induced paths of length 8 or more. We use a generic reduction from the precoloring extension node coloring problem

(in short **PREXT NODE COLORING**). This latter problem studied in [2, 10, 13, 11], can be described as follows. Given a positive integer k , a graph $G = (V, E)$ and k pairwise disjoint subsets V_1, \dots, V_k of V , we want to decide if there exists a node coloring $\mathcal{S} = (S_1, \dots, S_k)$ of G such that $V_i \subseteq S_i$, for all $i \leq k$. Moreover, we restrict to some class of graphs \mathcal{G} : we assume that \mathcal{G} is closed when we add a pending edge with a new node (i.e., if $G = (V, E) \in \mathcal{G}$ and $x \in V, y \notin V$, then $G + [x, y] \in \mathcal{G}$).

Theorem 2. \mathcal{G} **PREXT NODE COLORING** NP
 \mathcal{G} **MIN WEIGHTED NODE COLORING** NP \mathcal{G}

Let \mathcal{G} be such a class of graphs. We shall reduce **PREXT NODE COLORING** in \mathcal{G} graphs to weighted node coloring in \mathcal{G} graphs. Let $G = (V, E) \in \mathcal{G}$ and k pairwise disjoint subsets V_1, \dots, V_k of V . We build instance $I = (G', w)$ of weighted node coloring using several gadgets T_i , for $i = 1, \dots, k$. The construction of T_i is given by induction as follows: T_1 is simply a root v_1 with weight $w(v_1) = 2^{k-1}$. Given T_1, \dots, T_{i-1} , T_i is a tree with a root v_i of weight $w(v_i) = 2^{k-i}$ that we link to tree T_p via edge $[v_i, v_p]$ for each $p = 1, \dots, i - 1$.

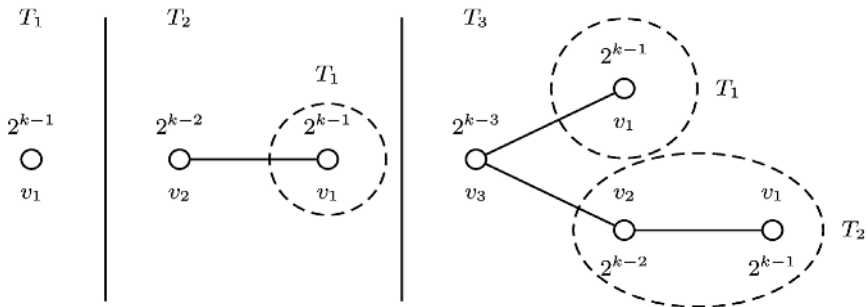


Fig. 4. Gadgets for T_1, T_2 and T_3

Figure 4 illustrates the gadgets T_1, T_2, T_3 . Now, $I = (G', w)$ where $G' = (V', E')$ is constructed in the following way: G' contains G . For all $i = 1, \dots, k$, we replace each node $v \in V_i$ by a copy of the gadget T_i where we identify v with root v_i . For all $v \in V \setminus (\cup_{i=1}^k V_i)$ we set $w(v) = 1$. Note that, by hypothesis, $G' \in \mathcal{G}$.

One can verify that the precoloring of G (given by V_1, \dots, V_k) can be extended to a proper node coloring of G' using at most k colors iff $opt(I) \leq 2^k - 1$.

Using the results of Kratochvil [13] on the **NP**-completeness of **PREXT NODE COLORING** in bipartite planar graphs for $k = 3$ and P_{13} -free bipartite graphs for $k = 5$, we deduce:

Corollary 1. $\frac{108}{7} - \varepsilon$ **MIN WEIGHTED NODE COLORING** NP $P=NP$

Corollary 2. P_{21} MIN WEIGHTED NODE COLORING
 NP $\frac{32}{31} - \varepsilon$ $P=NP$

In Hujter and Tuza [11], it is shown that PREXT NODE COLORING is **NP**-complete in P_6 -free bipartite chordal graphs for unbounded k . Unfortunately, we cannot use this result in Theorem 2 since the resulting graph has an induced path with arbitrarily large length. However, we can adapt their reduction.

Theorem 3. MIN WEIGHTED NODE COLORING NP P_8

We shall reduce 3-SAT-3, proved to be **NP**-complete in Papadimitriou [16] to our problem. Given a collection $\mathcal{C} = (C_1, \dots, C_m)$ of clauses over the set $X = \{x_1, \dots, x_n\}$ of Boolean variables such that each clause C_j has at most three literals and each variable has at most 3 occurrences (2 positive and one negative), we construct an instance $I = (BP, w)$ in the following way: we start from $BP_1 = (L_1, R_1; E_1)$, a complete bipartite graph $K_{n,m}$ where $L_1 = \{x_1, \dots, x_n\}$ and $R_1 = \{c_1, \dots, c_m\}$. Moreover, each node of BP_1 has weight 1. There is also another bipartite graph BP_2 isomorphic to $K_{2n,2n}$ where a perfect matching has been deleted. More formally, $BP_2 = (L_2, R_2; E_2)$ where $L_2 = \{l_1, \dots, l_{2n}\}$, $R_2 = \{r_1, \dots, r_{2n}\}$ and $[l_i, r_j] \in E_2$ iff $i \neq j$. Finally, $w(l_i) = w(r_i) = 2^{2n-i}$ for $i = 1, \dots, 2n$. Indeed, sets $\{l_{2i-1}, r_{2i-1}\}$ and $\{l_{2i}, r_{2i}\}$ will correspond to literal x_i and \bar{x}_i respectively. Between BP_1 and BP_2 , there is a set E_3 of edges. $[x_i, r_j] \notin E_3$ iff $j = 2i - 1$ or $j = 2i$ and $[l_i, c_j] \notin E_3$ iff $i = 2k - 1$ and x_k is in C_j or $i = 2k$ and \bar{x}_k is in C_j . Note that BP is a P_8 -free bipartite graph. One can verify that (X, \mathcal{C}) is satisfiable iff $opt(I) \leq 2^{2n} - 1$.

We end this section by stating that MIN WEIGHTED NODE COLORING is polynomial for P_5 -free bipartite graphs, i.e., without induced chain on 5 nodes. There are several characterizations of P_5 -free bipartite graphs, see for example, Hammer et al. [9], Chung et al. [3] and Hujter and Tuza [10]. In particular, BP is a P_5 -free bipartite graph iff BP is bipartite and each connected component of BP is $2K_2$ -free, i.e., its complement is C_4 -free. In this case, we can show that any optimal weighted node coloring $\mathcal{S}^* = (S_1^*, \dots, S_\ell^*)$ uses at most 3 colors (so, $\ell \leq 3$) and when $\ell = 3$, then for any connected component $BP_i = (L_i, R_i; E_i)$ of P_5 -free bipartite graph we have $S_1^{*,i} \cap L_i \neq \emptyset$ and $S_1^{*,i} \cap R_i \neq \emptyset$, $S_2^{*,i} \subset R_i$ (resp., $S_2^{*,i} \subset L_i$) and $S_3^{*,i} \subset L_i$ (resp., $S_3^{*,i} \subset R_i$) where $(S_1^{*,i}, S_2^{*,i}, S_3^{*,i})$ is the restriction of \mathcal{S}^* to the subgraph BP_i . Thus, applying an exhaustive search on $k_1 = w(S_2^*)$ and a dichotomy search $k_2 = w(S_3^*)$ we can find an optimal solution within $O(n|w|\log|w|)$ time where $|w| = |\{w(v) : v \in V\}|$. Hence, we can state:

Theorem 4. MIN WEIGHTED NODE COLORING P_5
 $O(n|w|\log|w|)$

3.2 Approximation

In Demange et al. [4], a $\frac{4}{3}$ -approximation is given for MIN WEIGHTED NODE COLORING and it is proved that a $(\frac{8}{7} - \varepsilon)$ -approximation is not possible, for any

$\epsilon > 0$, unless $\mathbf{P}=\mathbf{NP}$, even if we consider arbitrarily large values of $opt(I)$. Using Corollary 1, we deduce that this lower bound also holds if we consider bipartite planar graphs. Here, we give a $\frac{8}{7}$ -approximation in bipartite graphs.

BIPARTITECOLOR

- 1 Sort the nodes in non-increasing weight order (i.e., $w(v_1) \geq \dots \geq w(v_n)$);
- 2 For $i = 1$ to n do
 - 2.1 Set $V_i = \{v_1, \dots, v_i\}$;
 - 2.2 Compute $S_i^* = (S_1^i, S_2^i)$ (S_2^i may be empty) an optimal weighted node 2-coloring in the subgraph $BP[V_i]$ induced by V_i ;
 - 2.3 Define node coloring $S^i = (S_1^i, S_2^i, L \setminus V_i, R \setminus V_i)$ ($L \setminus V_i$ or/and $R \setminus V_i$ may be empty);
- 3 Output $S = argmin\{val(S^i) : i = 1, \dots, n\}$;

The step 2.2 consists of computing the (unique) 2-coloration $(S_{1,j}^*, S_{2,j}^*)$ (with $w(S_{1,j}^*) \geq w(S_{2,j}^*)$) of each connected component $BP_j, j = 1 \dots p$ of $BP[V_i]$ (with $S_{2,j}^* = \emptyset$ if BP_j is an isolated node). Then it merges the most expensive sets, i.e. it computes $S_1^i = \cup_{j=1}^p S_{i,j}^*$ for $i = 1, 2$. It is easy to observe that $S_i^* = (S_1^i, S_2^i)$ is the best weighted node coloring of $BP[V_i]$ among the colorings using at most 2 colors; such a coloring can be found in $O(m)$ time where $m = |E|$.

Theorem 5. *BIPARTITECOLOR*, $O(nm)$ MIN WEIGHTED NODE COLORING $\frac{8}{7}$

Let $I = (BP, w)$ be a weighted bipartite-graph where $BP = (L, R; E)$ and $S^* = (S_1^*, \dots, S_l^*)$ be an optimal node coloring of I with $w(S_1^*) \geq \dots \geq w(S_l^*)$. If $l < 3$, then BIPARTITECOLOR finds an optimal weighted node coloring which is S^n . Now, assume $l \geq 3$ and let $i_j = \min\{k : v_k \in S_j^*\}$. We have $i_1 = 1$ and $opt(I) \geq w(v_{i_1}) + w(v_{i_2}) + w(v_{i_3})$.

Let us examine several steps of this algorithm. When $i = i_2 - 1$, the algorithm produces a node 3-coloring $S^{i_2-1} = (S_{i_2-1}^1, L \setminus S_{i_2-1}^1, R \setminus S_{i_2-1}^1)$. Indeed, by construction $V_{i_2-1} \subseteq S_1^*$ is an independent set, and then, $S_{i_2-1}^*$ is defined by $S_{i_2-1}^{i_2-1} = V_{i_2-1}, S_{i_2-1}^{i_2-1} = \emptyset$ and then $val(S^{i_2-1}) \leq w(v_{i_1}) + 2w(v_{i_2})$. When $i = i_3 - 1$, the algorithm produces on $BP[V_{i_3-1}]$ a node 2-coloring $S_{i_3-1}^*$ with a cost $val(S_{i_3-1}^*) \leq w(v_{i_1}) + w(v_{i_2})$ since the coloring $(S_1^* \cap V_{i_3-1}, S_2^* \cap V_{i_3-1})$ is a feasible node 2-coloring of $BP[V_{i_3-1}]$ with cost $w(v_{i_1}) + w(v_{i_2})$. Thus, $val(S^{i_3-1}) \leq w(v_{i_1}) + w(v_{i_2}) + 2w(v_{i_3})$. Finally, when $i = n$, the node 2-coloring S^n satisfies $val(S^n) \leq 2w(v_{i_1})$

The convex combination of these 3 values with coefficients $\frac{1}{7} \times val(S^n), \frac{4}{7} \times val(S^{i_3-1})$ and $\frac{2}{7} \times val(S^{i_2-1})$ gives the expected result.

4 Weighted Edge Coloring in Bipartite Graphs

The weighted edge coloring problem on a graph G can be viewed as the weighted node coloring problem on $L(G)$ where $L(G)$ is the line graph of G . Here, for simplicity, we refer to the edge model.

4.1 Complexity Results

Demange et al. [4] have proved that MIN WEIGHTED EDGE COLORING in bipartite cubic graphs is strongly NP-complete and a lower bound of $\frac{8}{7}$ is given for the approximation. Here, we slightly improve these complexity results.

Theorem 6. $NP \leq_{\frac{7}{6} - \epsilon} MIN\ WEIGHTED\ EDGE\ COLORING \leq_{\epsilon} P=NP$

We shall reduce PREXT EDGE COLORING in bipartite cubic planar graphs to our problem. Given a bipartite cubic planar graph BP and 3 pairwise disjoint matchings E_i , the question of PREXT EDGE COLORING is to determine if it is possible to extend the edge precoloring E_1, E_2, E_3 to a proper 3-edge coloring of G . Very recently, this problem has been shown NP-complete in Marx [15]. Let $BP = (V, E)$ and E_1, E_2, E_3 be an instance of PREXT EDGE COLORING; we construct an instance $I = (BP', w)$ of weighted edge coloring as follows. Each edge in E_1 receives weight 3. Each edge $[x, y] \in E_2$ is replaced by a gadget F_2 described in Figure 4.1, where we identify x and y to v_0 and v_9 respectively. Each edge in E_3 is replaced by a gadget F_3 which is the same as gadget F_2 except that we have exchanged weights 1 and 2. The other edges of G receive weight 1. Remark that BP' is still a bipartite cubic planar graph.

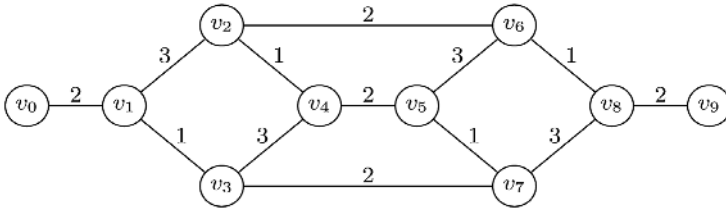


Fig. 5. Gadget F_2 for $e \in E_2$

We can verify that the answer of PREXT EDGE COLORING instance is yes if and only if there exists an edge coloring S of I with cost $val(S) \leq 6$.

4.2 Approximation

In Demange et al. [4], a $\frac{5}{3}$ -approximation is given for MIN WEIGHTED EDGE COLORING in bipartite graphs with maximum degree 3. Here, we give a $\frac{7}{6}$ -approximation. We need some notations: If $BP = (V, E)$ is a bipartite graph with node set $V = \{v_1, \dots, v_n\}$, we always assume that its edges $E = \{e_1, \dots, e_m\}$ are sorted in non-increasing weight order (i.e., $w(e_1) \geq \dots \geq w(e_m)$). If V' is a subset of nodes and E' a subset of edges, $BP[V']$ and $BP[E']$ denote the subgraph of BP induced by V' and the partial graph of BP induced by E' respectively. For any $i \leq m$, we set $E_i = \{e_1, \dots, e_i\}$ and $\overline{E}_i = E \setminus E_i$. Finally, V_i denotes the set of nodes of BP incident to an edge in E_i (so, it is the subset of non-isolated nodes of $BP[E_i]$).

BIPARTITEEDGECOLOR

- 1 For $i = m$ downto 1 do
 - 1.1 Apply algorithm **SOL1** on $BP[E_i]$;
 - 1.2 If $\text{SOL1}(BP[E_i]) \neq \emptyset$, complete in a greedy way all the colorings produced by **SOL1** on the edges of $\overline{E_i}$. Let $\mathcal{S}_{1,i}$ be a best one among these edge colorings of BP ;
 - 1.3 For $j = i$ downto 1 do
 - 1.3.1 Apply algorithm **SOL2** on $BP[E_j]$;
 - 1.3.2 If $\text{SOL2}(BP[E_j]) \neq \emptyset$, complete in a greedy way all the colorings produced by **SOL2** on the edges of $\overline{E_j}$. Let $\mathcal{S}_{2,j,i}$ be a best one among these edge colorings of BP ;
 - 1.3.3 Apply algorithm **SOL3** on $BP[E_j]$;
 - 1.3.4 If $\text{SOL3}(BP[E_j]) \neq \emptyset$, complete in a greedy way all the colorings produced by **SOL3** on the edges of $\overline{E_j}$. Let $\mathcal{S}_{3,j,i}$ be a best one among these edge colorings of BP
- 2 Output $\mathcal{S} = \text{argmin}\{val(\mathcal{S}_{1,i}), val(\mathcal{S}_{k,j,i}) : k = 2, 3, j = 1, \dots, i, i = 1, \dots, m\}$.

The greedy steps 1.2, 1.2.2 and 1.2.4 give a solution using at most 5 colors. More generally, in [4], we have proved that, in any graph G , the greedy coloring and at least one optimal weighted node coloring use at most $\Delta(G) + 1$ colors, where $\Delta(G)$ denotes the maximum degree of G . In our case, we have $G = L(H)$, the line graph of H , and we deduce $\Delta(L(H)) + 1 \leq 2(\Delta(H) - 1) + 1 = 2\Delta(H) - 1$. The 3 algorithms **SOL1**, **SOL2** and **SOL3** are used on several partial graphs BP' of BP . In the following, V' , E' and m' denote respectively the node set, the edge set and the number of edge of the current graph BP' . Moreover, we set $\overline{V'_i} = V' \setminus V'_i$. If $M = (M_1, \dots, M_\ell)$ with $w(M_1) \geq \dots \geq w(M_\ell)$ is an edge coloring of BP' , we note $i_j = \min\{k : e_k \in M_j\}$. We assume, for reason of readability, that some colors M_j may be empty (in this case $i_j = m' + 1$). The principle of these algorithms consist in finding a decomposition of BP' (a subgraph of BP) into two subgraphs BP'_1 and BP'_2 having each a maximum degree 2. When there exists such a decomposition, we can color BP'_i with at most 2 colors since BP is bipartite.

SOL1

- 1 For $j = m'$ downto 1 do
 - 1.1 If the degree of $BP'[E'_j]$ is at most 2 then
 - 1.1.1 Consider the graph BP'^j induced by the nodes of BP' incident to at least 2 edges of $\overline{E'_j}$ and restricted to the edges of $\overline{E'_j}$.
 - 1.1.2 Determine if there exists a matching M^j of BP'^j such that every node of $\overline{V'_j}$ is saturated;
 - 1.1.3 If such a matching is found, consider the decomposition $BP'_{1,j}$ and $BP'_{2,j}$ of BP' induced by $E'_j \cup M^j$ and $E' \setminus (E'_j \cup M^j)$ respectively;

- 1.1.4 Find an optimal 2-edge coloring (M_1^j, M_2^j) of $BP'_{1,j}$;
 - 1.1.5 Color greedily the edges of $BP'_{2,j}$ with two colors (M_3^j, M_4^j) ;
 - 1.1.6 Define $\mathcal{S}_1^j = (M_1^j, M_2^j, M_3^j, M_4^j)$ the edge coloring of BP' ;
- 2 Output $\{\mathcal{S}_1^j : j = 1, \dots, m' - 1\}$;

Note that the step 1.1.2 is polynomial. Indeed, more generally, given a graph G and $V' \subseteq V$, it is polynomial to determine if there exists a matching such that each node of V' is matched. To see this, consider G' where we add to G all missing edges between nodes of $V \setminus V'$. If $|V|$ is odd, then we add a node to the clique $V \setminus V'$. It is easy to see that G' has a perfect matching if and only if G has a matching such that each node of V' is saturated.

Lemma 1. $\mathcal{S} = (M_1, M_2, M_3, M_4)$... BP' ... \bullet ...
 $val(\mathcal{S}_1^{i_3-1}) \leq w(M_1) + w(M_2) + 2w(M_3)$

SOL2

- 1 For $k = m'$ downto 1 do
 - 1.1 If E'_k is a matching :
 - 1.1.1 Determine if there exists a matching M_k of $BP'[\overline{V'_k}]$ such that each node of $BP'[\overline{V'_k}]$ having a degree 3 in BP' is saturated.
 - 1.1.2 If such a matching is found, consider the decomposition $BP'_{1,k}$ and $BP'_{2,k}$ of BP' induced by $E'_k \cup M_k$ and $E' \setminus (E'_k \cup M_k)$ respectively;
 - 1.1.3 Color $BP'_{1,k}$ with one color M_1^k ;
 - 1.1.4 Color greedily $BP'_{2,k}$ with two colors M_2^k and M_3^k ;
 - 1.1.5 Define $\mathcal{S}_2^k = (M_1^k, M_2^k, M_3^k)$ the edge coloring of BP' ;
- 2 Output $\{\mathcal{S}_2^k : k = 1, \dots, m'\}$;

Lemma 2. $\mathcal{S} = (M_1, M_2, M_3)$... BP' ... \bullet ...
 $val(\mathcal{S}_2^{i_2-1}) \leq w(M_1) + 2w(M_2)$

SOL3

- 1 For $k = m'$ downto 1 do
 - 1.1 Determine if there is a matching M_k in $BP'[\overline{E'_k}]$ such that each node of degree 3 in BP' is saturated.
 - 1.2 If such a matching is found, consider the decomposition $BP'_{1,k}$ and $BP'_{2,k}$ of BP' induced by M_k and $E' \setminus M_k$ respectively;
 - 1.3 Color $BP'_{1,k}$ with one color M_3^k ;
 - 1.4 Color greedily $BP'_{2,k}$ with two colors M_1^k and M_2^k ;
 - 1.5 Define $\mathcal{S}_3^k = (M_1^k, M_2^k, M_3^k)$ the edge coloring of BP' ;
- 2 Output $\{\mathcal{S}_3^k : k = 1, \dots, m' - 1\}$;

Lemma 3. $\mathcal{S} = (M_1, M_2, M_3) \dots BP'$
 $val(\mathcal{S}_3^{i_3-1}) \leq 2w(M_1) + w(M_3)$

Theorem 7. BIPARTITEEDGECOLOR $\frac{7}{6}$ MIN WEIGHTED EDGE COLORING

Let $\mathcal{S}^* = (M_1^*, \dots, M_5^*)$ with $w(M_1^*) \geq \dots \geq w(M_5^*)$ be an optimal weighted edge coloring of BP . Denote by i_k^* the smallest index of an edge in M_k^* ($i_k^* = m+1$ if the color is empty). Consider the iteration of BIPARTITEEDGECOLOR corresponding to the cases $i = i_5^* - 1$ and $j = i_4^* - 1$. Then, applying Lemma 1, we produce on $BP' = BP[E_i]$ an edge coloring of weight at most $w(M_1^*) + w(M_2^*) + 2w(M_3^*)$. Then the greedy coloring of the edges of $\overline{E_i}$ produces a coloring \mathcal{S}'_1 of weight $val(\mathcal{S}'_1) \leq w(M_1^*) + w(M_2^*) + 2w(M_3^*) + w(M_5^*)$. Applying the same arguments on Lemma 2 and Lemma 3, we produce two solutions \mathcal{S}'_2 and \mathcal{S}'_3 respectively satisfying $val(\mathcal{S}'_2) \leq w(M_1^*) + 2w(M_2^*) + 2w(M_4^*)$ and $val(\mathcal{S}'_3) \leq 2w(M_1^*) + w(M_3^*) + 2w(M_4^*)$.

Notice that if there is an empty color produced by one of the algorithms SOL i , then the bounds are still valid. The convex combination of these 3 values with coefficients $\frac{3}{6} \times val(\mathcal{S}'_1)$, $\frac{2}{6} \times val(\mathcal{S}'_2)$ and $\frac{1}{6} \times val(\mathcal{S}'_3)$ gives the expected result.

5 Weighted Node Coloring in Split Graphs

The split graphs are a class of graphs related to bipartite graphs. Formally, $G = (K_1, V_2; E)$ is a split graph if K_1 is a clique of G with size $|K_1| = n_1$ and V_2 is an independent set with size $|V_2| = n_2$. So, a split graph can be viewed as a bipartite graph where the left set is a clique. Since split graphs forms a subclass of perfect graphs, the node coloring problem on split graphs is polynomial. On the other hand, in [4], it is proved that the weighted node coloring problem is strongly NP-complete in split graphs, even if the weights take only two values. Thus, we deduce that there is no fully polynomial time approximation scheme in such a class of graphs. Here, we propose a polynomial time approximation scheme using structural properties of optimal solutions. An immediate observation of split graphs is that any optimal node coloring $\mathcal{S}^* = (S_1^*, \dots, S_\ell^*)$ satisfies $|K_1| \leq \ell \leq |K_1| + 1$ and any color S_i^* is a subset of V_2 with possibly one node of K_1 . In particular, for any optimal node coloring $\mathcal{S}^* = (S_1^*, \dots, S_\ell^*)$, there exists at most one index $i(\mathcal{S}^*)$ such that $S_{i(\mathcal{S}^*)}^* \cap K_1 = \emptyset$.

Lemma 4. $\mathcal{S}^* = (S_1^*, \dots, S_\ell^*)$
 $w(S_1^*) \geq \dots \geq w(S_\ell^*) \quad i_0 \leq \ell + 1$

- $\forall j < i_0 \ S_j^* = \{v_j\} \cup \{v \in V_2 : v \notin \cup_{k=1}^{j-1} S_k^* \text{ and } [v, v_j] \notin E\} \quad v_j \in K_1$
- $S_{i_0}^* = V_2 \setminus (S_1^* \cup \dots \cup S_{i_0-1}^*) \quad \forall j > i_0 \ S_j^* = \{v_j\} \quad v_j \in K_1$

Thus, applying an exhaustive search on all sets $K'_1 \subseteq K_1$ with $k = |K'_1| \leq \lceil \frac{1}{\epsilon} \rceil$ and on all bijections from $\{1, \dots, k\}$ to K'_1 , one can find the k heaviest colors of an optimal weighted node coloring and thus, we deduce:

Theorem 8. MIN WEIGHTED NODE COLORING

References

1. C. BERGE[1973]. Graphs and hypergraphs. *North Holland, Amsterdam*.
2. H. L. BODLAENDER, K. JANSEN, AND G. J. WOEGINGER[1990]. Scheduling with incompatible jobs. *Discrete Appl. Math.*, 55:219–232.
3. F. R. K. CHUNG, A. GYÁRFÁS, ZS. TUZA AND W. T. TROTTER [1990]. The maximum number of edges in 2K2-free graphs of bounded degree. *Discrete Mathematics*, 81:129–135.
4. M. DEMANGE, D. DE WERRA, J. MONNOT AND V.TH. PASCHOS [2002]. Weighted node coloring: when stable sets are expensive. *Proc. WG'02 LNCS* 2573:114–125.
5. M. R. GAREY AND D. S. JOHNSON [1979]. Computers and intractability. a guide to the theory of NP-completeness. *CA, Freeman*.
6. H. GROTZSCH [1959]. Ein dreifarbensatz für dreikreisfreie netze auf der Kugel. *Wiss. Z. Martin Luther Univ. Halle-Wittenberg, Math. Naturwiss Reihe*, 8:109–120.
7. D. J. GUAN AND X. ZHU [1997]. A Coloring Problem for Weighted Graphs. *Inf. Process. Lett.*, 61(2):77–81.
8. R. HASSIN AND J. MONNOT [2004]. The maximum saving partition problem. *Op. Res. Lett.*, to appear.
9. P. L. HAMMER, U. N. PELED AND X. SUN [1990]. Difference graphs. *Discrete Applied Mathematics*, 28:35–44.
10. M. HUJTER AND ZS. TUZA [1993]. Precoloring extension. II. Graphs classes related to bipartite graphs. *Acta Math. Univ. Comeniane*, LXII:1–11.
11. M. HUJTER AND ZS. TUZA [1996]. Precoloring extension. III. Classes of perfect graphs. *Combin. Probab. Comput.*, 5:35–56.
12. D. KÖNIG [1916]. über graphen und ihrer anwendung auf determinantentheorie und mengenlehre. *Math. Ann.*, 77:453–465.
13. J. KRATOCHVIL [1993]. Precoloring extension with fixed color bound. *Acta Math. Univ. Comen.*, 62:139–153.
14. D. LICHTENSTEIN [1982]. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343.
15. D. MARX [2004]. NP-completeness of list coloring and precoloring extension on the edges of planar graphs. *Technical report* available to <http://www.cs.bme.hu/~dmarx/publications.html>.
16. C. H. PAPADIMITRIOU[1994]. Computational Complexity. *Addison Wesley*.

Sweeping Graphs with Large Clique Number (Extended Abstract)

Boting Yang¹, Danny Dyer², and Brian Alspach³

¹ Department of Computer Science, University of Regina
boting@cs.uregina.ca

² Department of Mathematics, Simon Fraser University
tddyer@sfu.ca

³ Department of Mathematics and Statistics, University of Regina
alspach@math.uregina.ca

Abstract. An open problem in sweeping is the existence of a graph with connected sweep number strictly less than monotonic connected sweep number. We solve this problem by constructing a graph W exhibiting exactly this property. Further, we will examine a new method of constructing graphs that makes proving all such inequalities easier, and offer some new lower bounds on sweep numbers.

1 Introduction

The model of searching originated by Parsons in [6] allowed capture of an intruder by a searcher to occur on an edge. We will call this type of search a *sweep*. The specifics of sweeping a graph G are as follows: initially, all edges of G are *uncleared* (or *dirty*). To sweep G it is necessary to formulate and carry out a sweep strategy. A *sweep strategy* is a sequence of actions designed so that the final action leaves all edges of G *cleared* (or *clean*). The actions allowed after placing all the sweepers on vertices are: move a single sweeper along an edge uv starting at u and ending at v ; move a sweeper on a vertex u to any other vertex v .

Any sweep strategy that uses the second action will be called a *vertex sweep strategy*. We will restrict ourselves to considering simple graphs throughout this paper.

An edge uv in G can be cleared in one of two ways: (1) At least two sweepers are placed on vertex u and one of them traverses the edge from u to v while the others remain on u . (2) At least one sweeper is placed on vertex u , and all other edges incident with u , other than uv , are already cleared. Then one sweeper moves from u to v .

Knowing that our goal is to end up with a graph where all the edges are cleared, a basic question is: what is the fewest number of sweepers for which a sweep strategy exists for G ? We call this the *sweep number*, denoted $sw(G)$. Similarly, we define the *vertex sweep number*, denoted $vsw(G)$. In fact, these

two numbers are equal for connected graphs [1]. This paper will deal solely with connected graphs.

Further restrictions may be placed on sweep strategies. Let $E(i)$ be the set of cleared edges after action i has occurred. A sweep strategy for a graph G for which $E(i) \subseteq E(i+1)$ for all i is said to be *connected*. We then define the *connected sweep number*, $\text{msw}(G)$. Similarly, a sweep strategy such that $E(i)$ induces a connected subgraph for all i is said to be *monotonic*, and we define the *monotonic sweep number*, $\text{ksw}(G)$. Finally, a sweep strategy can be both connected and monotonic, giving us the *monotonic connected sweep number*, $\text{mksw}(G)$. We similarly define the same concepts for wormhole sweeping.

LaPaugh [4] and Bienstock and Seymour [3] proved that for any connected graph G , $\text{wsw}(G) = \text{mmsw}(G)$. Barrière et al. [2] extended this result, giving the following relations for these numbers.

Lemma 1. For any graph G , $\text{wsw}(G) = \text{sw}(G) = \text{mmsw}(G) \leq \text{msw}(G) \leq \text{kmsw}(G) = \text{ksw}(G) \leq \text{mkmsw}(G) = \text{mksw}(G)$.

We will show that $\text{sw}(K_n) = \text{mksw}(K_n) = n$, where K_n is the complete graph on n vertices. This means that there is exactly one sweep number for complete graphs. More generally, determining the sweep number of a graph G is NP-complete [5]. As any successful sweep strategy gives an upper bound, our goal becomes first to find the “right” way to sweep the graph, using as few sweepers as possible. However, once this strategy is found, we must then prove that no fewer sweepers will suffice. Here is where the true difficulty lies: most easily attainable lower bounds we have are quite poor. We will prove several lower bound results, but the one we will make the greatest use of is the clique number, which occurs repeatedly in our constructions. The *clique number* is the number of vertices in the largest clique of G , denoted $\omega(G)$.

Returning to Lemma 1, we still need to consider the question of strict inequality. An example given in [2] shows that the first inequality can be strict. The second inequality, $\text{msw} \leq \text{ksw}$, was also proved in [2]. They also gave an example showing that the inequality was strict. With this result, they also observed that the classes of monotonically sweepable and connected sweepable graphs were not minor closed. We prove these results by using large cliques as our building blocks, thereby allowing us to calculate the sweep numbers easily.

Whether the third inequality, $\text{ksw} \leq \text{mksw}$, could be strict was an open problem [2]. We will show that there exists a graph W with $\text{ksw}(W) < \text{mksw}(W)$, and that the difference between these two values can be arbitrarily large.

This paper is organized as follows: in Section 2, we examine the sweeping of cliques, and some of the immediate consequences. Section 3 describes the construction of a graph W such that $\text{ksw}(W) < \text{mksw}(W)$, and includes some lemmas that hint at the construction of the graph. Sections 4 and 5 are devoted to the analysis of the connected sweep number and monotonic connected sweep number of W . Section 6 re-examines several of the observations of [2], proving not only that inequalities can be strict, but the difference can be arbitrarily large. The proofs of these results rely heavily on the idea of clique number. Finally, we mention several open problems in Section 7.

2 Sweeping and Cliques

The main result of this section is Theorem 1. We use it to prove several lower bounds for the sweep number.

It is easy to see that $\text{sw}(K_1) = 1$, $\text{sw}(K_2) = 1$, $\text{sw}(K_3) = 2$, and $\text{sw}(K_4) = 4$. The jump of the sweep numbers from 2 for K_3 to 4 for K_4 indicates that obvious methods, such as mathematical induction, will not easily prove a formula for $\text{sw}(K_n)$. But the following result will give $\text{sw}(K_n)$ as a corollary.

Theorem 1. *Let G be a graph with minimum degree $\delta(G) \geq 3$. Then $\text{sw}(G) \geq \delta(G) + 1$.*

Consider a graph G with minimum degree $\delta(G) = \delta$, and a strategy S that sweeps it. A vertex v is said to be *clean* if all the edges incident with it are currently uncontaminated. If the first vertex cleared by S is not of minimum degree, then it must have at least $\delta + 1$ vertices adjacent to it. When this vertex is cleared, each of these vertices must contain a sweeper and $\text{sw}(G) \geq \delta + 1$.

We now consider the last time that the graph goes from have no cleared vertices to a single cleared vertex u . By the preceding paragraph, we may assume u is a vertex of minimum degree. We will assume that the strategy S employs at most δ sweepers, and arrive at a contradiction. Let the neighbours of u be denoted $v_1, v_2, \dots, v_\delta$. Assume, without loss of generality, that uv_1 is the final edge incident with u cleared, and that uv_2 is the penultimate such edge.

Consider the placement of sweepers the moment before uv_1 is cleared. As each of uv_i ($2 \leq i \leq \delta$) is cleared, there must be a sweeper on each endpoint of these edges. But this uses all δ sweepers, so the only way that uv_1 can be cleared is if the sweeper at u sweeps uv_1 to end up at v_1 . Thus, all other edges incident with v_1 must be contaminated. Since $\delta \geq 3$, the sweeper on v_1 cannot move.

Now consider the placement of sweepers before the penultimate edge uv_2 is cleared. As each of the edges uv_i ($3 \leq i \leq \delta$) is cleared, there must be a sweeper on each endpoint of these edges, accounting for $\delta - 1$ sweepers. Since the next move is to clear uv_2 , the single free sweeper must be on either u or v_2 . Moving from v_2 to u would not sweep the edge. Instead, the edge must be cleared from u to v_2 . This leaves the sweeper at v_2 , and all the other edges incident with v_2 must be contaminated. Since $\delta \geq 3$, the sweeper on v_2 cannot move.

Consider a sweeper on v_i , $3 \leq i \leq \delta$. If the vertex v_i is adjacent to v_1 and v_2 , then the edges v_1v_i and v_2v_i are contaminated, and the sweeper at v_i cannot move.

If the vertex v_i is adjacent to exactly one of v_1 and v_2 , it must also be adjacent to some other vertex, w , not adjacent to u (as the degree of v_i is at least δ). As there is no sweeper on w , the only way that v_iw can be cleared is if w is a cleared vertex. This contradicts the choice of u . Thus, the sweeper at v_i cannot move.

Finally, if the vertex v_i is adjacent to neither v_1 nor v_2 , it must be adjacent to two vertices, w_1 and w_2 , neither of which is adjacent to u . As before, these edges cannot be clean, and thus the sweeper at v_i cannot move. As there are still

contaminated edges, but none of the δ sweepers can move, we have obtained the required contradiction. ■

Corollary 1. $G, \kappa(G) \geq 3, \kappa'(G) \geq \kappa(G) + 1$
 $sw(G) \geq \kappa'(G) + 1 \geq \kappa(G) + 1$

Corollary 2. $n \geq 4, sw(K_n) = n$

Corollary 2 can actually be strengthened to $mksw(K_n) \leq n$. From Lemma 1 and Corollary 2, it follows that there is only one sweep number for the complete graph K_n .

Corollary 3. $n, wsw(K_n) = sw(K_n) = msw(K_n) = msw(K_n) = ksw(K_n) = ksw(K_n) = mksw(K_n) = mksw(K_n) = n$.

Theorem 2. $G', G, sw(G') \leq sw(G)$

Theorem 3. $n \geq 4, K_{n+1}, K_n, n + 1$

Theorem 4. $n \geq 4, n, n - 1, K_n$

We remind the reader that $G \square H$ denotes the cartesian product of two graphs G and H .

Theorem 5. $n \geq 4, mksw(K_n \square K_n) \leq n(n - 1) + 1$

3 Construction of the Obstruction W

From Corollary 3 and Theorem 2, we obtain the following result.

Lemma 2. $G, \omega(G) \geq 4, \omega(G) \leq sw(G)$

Lemma 2 is a useful lower bound and a hint as to how to construct graphs to calculate sweep number easily. Given a graph G that can be cleared by p sweepers, we can instead construct a graph G' , almost identical to G , with the addition of a K_p to “force” the sweep number of G' to be at least p .

Theorem 6. $G, n, sw(G) = n + 1, G = K_n \square K_2$

Theorem 6 also provides hints about constructing graphs with easily calculated sweep numbers. In particular, if two large cliques are interconnected too heavily, the sweep number goes up. If we again consider a graph G which is p -sweepable, we have already mentioned that in a new graph G' , we can preserve the structure of G but add a K_p , to ease calculating the sweep number. However, if many large cliques are added that are too heavily interconnected to one another, G' will no longer be p -sweepable.

We construct the graph W as shown in Figure 1. In this figure, a circle represents a complete graph on the indicated number of vertices, and double lines between two cliques A and B indicate a perfect matching either between A

Similarly, we sweep the D'_i , E'_i , and F'_i , ending with sweepers on $A'_4[D'_1]$, $A'_5[F'_1]$, and $A'_6[E'_1]$. We now have 201 free sweepers.

We send these sweepers to a single vertex in A'_8 that is adjacent to $A'_6[E'_1]$ and clear this vertex. This leaves a single free sweeper that cleans all remaining edges in A'_8 . Then, the sweepers in $A'_8[A'_6]$ move to A'_6 , and a single free sweeper cleans all edges in A'_6 .

At this point, we have 20 sweepers stationed on each of $A'_2[C'_1]$, $A'_4[D'_1]$, and $A'_5[F'_1]$, and 80 stationed on A'_6 , leaving 141 free sweepers. Move all these sweepers to a vertex of A'_7 that is adjacent to $A'_5[F'_1]$. We then clear this vertex, and with the remaining free sweeper, clean the edges of A'_7 . Lifting the sweepers on $A'_7[A'_5]$ along the perfect matching to A'_5 , clean A'_5 . We now have 80 sweepers station on A'_5 , 80 on A'_6 , 20 on $A'_2[C'_1]$ and 20 on $A'_4[D'_1]$, leaving 81 free sweepers.

Move along the perfect matchings in the T'_i , using the single free sweeper to clean all the edges in the T'_i , and finally sweep from T'_1 to A'_6 . We now can move 160 sweepers from A'_6 to L'_{300} . Use 1 sweeper to clean all the edges in L'_{300} . Then sweep to L'_{299} , and use the single free sweeper to clean the edges there, and continue until the edges in L'_1 are cleared. Then, send the sweepers from $L'_1[A'_2]$ to A'_2 , and use a free sweeper to clean all the edges in A'_2 . There are now 80 sweepers stationed at each of A'_2 and A'_5 , and 20 stationed on $A'_4[D'_1]$, leaving 101 free sweepers.

As we cleared the L'_i , we also clear all the R'_i , using the 101 free sweepers with those 80 sweepers on A'_5 . Then we move the sweepers from $R'_1[A'_4]$ to A'_4 , and use a single sweeper to clean the remaining edges of A'_4 . We now have 190 sweepers stationed at A'_4 and A'_2 , leaving 91 free sweepers. We use these sweepers to clean the B'_i , and then we use 160 sweepers to clean A'_3 . This cleans the left half of the graph W . We now only have 80 sweepers stationed at A'_2 .

We now use 281 sweepers to clean, one by one, the 300 cliques between A'_1 and A_1 , followed by A_1 itself. Then, we move the sweepers from $A_1[A_2]$ to A_2 , and use a free sweeper to clean all edges in A_2 . We now have 80 sweepers stationed in A_2 , leaving 201 free sweepers.

Now pick a vertex in C_1 . Sweep to this vertex from $A_2[C_1]$. Then move another sweeper along this edge, and to the corresponding vertex in C_2 , then another to the corresponding vertex in C_3 , and so on, until finally we have placed a sweeper on the corresponding vertex in $A_9[C_{19}]$. Then, move a sweeper to a vertex in $A_9[D_{19}]$, then move a sweeper to a corresponding vertex in D_{19} , then another to a corresponding vertex in D_{18} , and so on, until sweeping to the corresponding vertex in D_1 . Finally, move one sweeper to the corresponding vertex in $A_4[D_1]$. We now have 80 sweepers stationed in A_2 , and in total, 41 sweepers along a path through the C_i , through A_9 , and finally through the D_i into A_4 . This leaves 160 free sweepers.

Move these free sweepers along this path, to the single vertex in A_3 adjacent to the path. Clear this vertex, and then use the single free sweeper to clean A_3 . Then the sweepers on $A_3[A_4]$ sweep to A_4 , and a free sweeper clean A_4 . With 110 sweepers stationed on A_4 , 80 sweepers stationed on A_2 , and 39 sweepers

strung in that path from A_1 to A_4 through A_9 , this leaves 52 free sweepers. These sweepers can clean the B_i .

Now surrender the path from A_2 to A_4 through A_9 in the following manner: the single sweeper in D_1 moves to the corresponding vertex in D_2 ; the two sweepers in D_2 move to the corresponding vertex in D_3 ; and so on, until finally all the sweepers in the path are in C_1 , at which point they move to $A_2[C_1]$. (“Reeling in” the path in this manner preserves the connectedness of this sweep.)

We now have 190 sweepers stationed on A_2 and A_4 , leaving 91 sweepers free. Of these free sweepers, move 20 from A_4 to D_1 , using a free sweeper to clean D_1 , and then clean D_2 , then D_3 , until finally we station 20 sweepers on $A_9[D_{19}]$. Now move the 110 sweepers on A_4 to $R_1[A_4]$. Use a free sweeper to clean the edges of $R_1[A_4]$. Place the remaining 71 free sweepers on a vertex on $R_1[A_4]$ and clear it. With the single remaining free sweeper, clean R_1 . Then sweep all sweepers in R_1 to R_2 , and use the remaining free sweeper to clean R_2 . Repeating, clean to R_{300} , finally moving 80 sweepers from $R_{300}[A_5]$ to A_5 and using a single free sweeper to clean A_5 . We now have 160 sweepers stationed at A_2 and A_5 , and 20 sweepers stationed at $A_9[D_{19}]$. This leaves 101 free sweepers.

As with the D_i , use 21 sweepers to clean the C_i , eventually stationing 20 sweepers at $A_9[C_{19}]$. Then sweep the 80 sweepers from A_2 to $L_1[A_2]$, and use the 81 free sweepers to first clear a vertex in $L_1[A_2]$ and then to clean L_1 . We then clean all the L_i , eventually stationing 80 sweepers at A_6 and using a free sweeper to clean the edges of A_6 . We now have 80 sweepers at each of A_5 and A_6 , and 20 sweepers at each of $A_9[C_{19}]$ and $A_9[D_{19}]$, leaving 81 free sweepers.

We use these sweepers to clean the T_i . Then, we sweep them along the F_i , eventually stationing 20 sweepers on $A_9[F_{300}]$. With 80 sweepers at A_5 , 80 at A_6 , and 60 in A_9 , there are 61 free sweepers. Move the 80 sweepers at A_5 and 61 free sweepers to A_7 , and clean A_7 .

Clean all the E_i by sweeping from A_6 , eventually stationing 20 sweepers at $A_9[E_{300}]$. Move the 80 sweepers stationed at A_6 to $A_8[A_6]$, using a free sweeper to clean the edges of $A_8[A_6]$. We now have 80 sweepers stationed in A_9 , and 80 sweepers stationed in $A_8[A_6]$. The remaining 121 sweepers are used to clear a vertex in $A_8[A_6]$, and then the 1 remaining free sweeper cleans the edges of A_8 .

Finally, with only 80 sweepers stationed in A_9 (and thus 201 free sweepers), we clear a vertex in A_9 , and then use the single remaining free sweeper to clean the edges of A_9 , which completes the sweep strategy for W . This strategy, as has been noted, is connected, but is not monotonic, as the edges in the path from A_2 through A_9 to A_4 were allowed to be recontaminated. ■

5 Computing $\text{mksw}(W)$

From the symmetry of W and Lemma 3, we can suppose that any monotonic connected sweep strategy first clears A'_9 and last clears A_9 .

Theorem 8. $\text{mksw}(W) = 290$

First, we show W is 290-monotonically connected sweepable. Starting at A'_9 , we sweep as in Theorem 7 until we clean all the edges in A_2 . At this point, we have 80 sweepers stationed at A_2 , and 210 free sweepers. Moving 50 of these sweepers to B_1 , we then use another free sweeper to clean the edges in B_1 . Then, repeat with B_2, B_3, \dots, B_{300} . Finally, station 50 sweepers at $A_4[B_{300}]$, and use another sweeper to clear all the edges in $A_4[B_{300}]$. We now have stationed 130 sweepers, and have 160 free.

Send these 160 sweepers to a single vertex in A_3 that is adjacent to $A_4[B_{300}]$, and clear this vertex. Using the 1 remaining free sweeper, clean the edges of A_3 . Then, move the sweepers from $A_3[A_4]$ to A_4 , and clean all remaining edges in A_4 . We now have 110 sweepers stationed at A_4 , and 80 stationed at A_2 , leaving 100 free sweepers. Use these sweepers to clean the D_i , eventually placing 20 sweepers on $A_9[D_{19}]$. Then move all the sweepers on A_4 to $R_1[A_4]$. We have now stationed 80 sweepers on A_2 , 20 sweepers on $A_9[D_{19}]$, and 110 on $R_1[A_4]$, leaving 80 free sweepers.

With 80 free sweepers and the 110 sweepers in R_1 , clean the R_i , eventually stationing 80 at A_5 , and use another sweeper to clean the edges of A_5 . We now have 80 sweepers stationed at each of A_2 and A_5 , and 20 stationed at $A_9[D_{19}]$, leaving 110 free sweepers.

Using these sweepers, clean the C_i , eventually stationing 20 at $A_9[C_{19}]$. We now have 90 free sweepers. Using these sweepers, and those from A_2 , we clean the L_i , eventually stationing 80 sweepers on A_6 . With 80 sweepers stationed on each of A_6 and A_5 , and 20 stationed on each of $A_9[C_{19}]$ and $A_9[D_{19}]$, we have 90 free sweepers to clean the T_i . Then, use these sweepers to clean the F_i , eventually stationing 20 sweepers on $A_9[F_{300}]$.

With 60 sweepers stationed in A_9 , and 80 stationed at A_6 , we have 150 (including 80 at A_5) to clean A_7 . Then, these sweepers can sweep the E_i , finally stationing 20 sweepers at $A_9[E_{300}]$. With 80 sweepers in A_9 , we have 210 sweepers (including 80 at A_6) to sweep A_8 . Finally, these 210 sweepers move to A_9 and clean it. Thus $\text{mksw}(W) \leq 290$.

To prove the equality, we will show that $\text{mksw}(W) > 289$. First, assume that W is 289-monotonically connected sweepable. Let S be a monotonic connected sweep strategy using 289 sweepers. Due to the 300 280-cliques between A_1 and A'_1 , it is clear that we must clean one ‘‘side’’ of W first, then proceed through the 280-cliques one at a time. Assume that the left side of W has been cleared first, and that now 80 sweepers are stationed at A_2 . We call a clique pseudo-cleared if it contains exactly one cleared vertex.

Case 1: If A_8 is pseudo-cleared before A_3 , then at the moment that the first vertex v in A_8 is cleared, there must be at least 200 sweepers in A_8 and 1 sweeper in A_6 . We now consider several subcases.

Case 1a: If the L_i are cleared before v is cleared, there must be at least 80 sweepers on A_6 . Since A_9 is not cleared, there must be at least 20 sweepers between A_2 and $A_9[C_{19}]$. With at least 200 sweepers in A_8 , this strategy requires at least 300 sweepers, a contradiction. Similarly, it can be shown that if the L_i

are not cleared before v is cleared, then none of the T_i , R_i , or B_i can be cleared before v is cleared.

Case 1b: If none of the L_i , R_i , T_i or B_i are cleared before v is cleared, then there must be at least 80 sweepers on A_2 . Further, since the strategy is monotonic and connected, there must be a cleared path from A_2 to A_8 that passes through A_9 before v is cleared. If the strategy passes through the C_i 's by leaving a single sweeper in each clique ("sneaking" through the cliques, but not cleaning them), we must station at least 19 sweepers on this sneaking path. On the other hand, if we clean each of the C_i clique by clique, we must station at least 20 sweepers on $A_9[C_{19}]$. In either situation, since there are at least 200 sweepers in A_8 and 1 on A_6 , this strategy must use at least 290 sweepers, a contradiction.

Case 2: If A_3 is pseudo-cleared before A_8 , then at the moment that the first vertex, u in A_3 is cleared there must be at least 159 sweepers on A_3 , and at least 1 sweeper on A_4 . Since A_8 is not cleared and A_2 is, there must be at least 80 sweepers somewhere between A_2 and A_8 .

Case 2a: If the B_i are cleared when u is cleared, there must be at least 49 more sweepers on A_4 . With at least 159 on A_3 , at least 50 on A_4 , at least 80 sweepers between A_2 and A_8 , this accounts for at least 289 sweepers, with no free sweeper to clean edges in A_3 , a contradiction. Similarly, it can be shown that none of the L_i , R_i , or T_i can be cleared when u is cleared.

Case 2b: If none of the L_i , B_i , T_i , or R_i are cleared when u is cleared, we must station at least 80 sweepers on A_2 . Further, since this strategy is monotonic and connected, there must be a cleared path from A_2 to A_3 that passes through A_9 when u has been cleared. We consider some subcases:

Case 2b(i): If the cleared path through the D_i is obtained by sweeping the D_i clique by clique, there must be at least 20 sweepers left at $A_9[D_{19}]$ and 19 more sweepers left at $A_4[D_1]$. Also, since the cleared path must go through the C_i , there must be at least 19 sweepers stationed on the C_i (or on $A_9[C_{19}]$). Then with at least 159 on A_3 , at least 20 on A_4 , and at least 80 on A_2 , this strategy uses at least 298 sweepers, a contradiction.

Case 2b(ii): If the cleared path through the D_i is obtained by placing only a single sweeper on each clique D_i , and a single sweeper on $A_9[D_{19}]$, this requires at most 20 sweepers. Whether the C_i are cleaned clique by clique, or merely have a single sweeper in each clique, this requires at most 20 sweepers (either 20 on $A_9[C_{19}]$ or 19 on the C_i and 1 on $A_9[C_{19}]$). With exactly 159 on A_3 , exactly 1 on A_4 , and exactly 80 on A_2 , this leaves 9 free sweepers. These sweepers can be used to clean A_3 , and then 109 more sweepers (for a total of 110) are stationed on A_4 . With 110 on A_4 , 80 on A_2 , and 40 through the C_i and D_i , this leaves 59 free sweepers with which we can sweep the B_i .

At this point, there are not enough free sweepers to clean either the L_i or R_i . The 59 sweepers are sufficient to clean either the E_i or the F_i , but not both. Then, no more cliques may be cleared. So, there is no monotonic connected sweep strategy for this graph using 289 sweepers. ■

Corollary 4. W , $ksw(W) = 281 < 290 = mksw(W)$

6 Applications of the Clique Method

6.1 Inequalities

We have exhibited the power of cliques in constructing the graph W . We can extend this technique to other graphs to study other properties of sweeping.

Let $X' = K_{10} \square P_{60}$. We construct X as in Figure 2, where each circle represents a complete graph on the number of vertices in the circle, and double lines between two cliques represent a saturated matching. By construction, $V(X_{21}[A_{20}] \cap V(X_{21}[B_{20}]) = \emptyset$ and $V(X_{40}[C_{41}] \cap V(X_{40}[D_{41}]) = \emptyset$. Then X is a subgraph of X' . However, we can prove $\text{ksw}(X) > \text{ksw}(X')$.

Theorem 9. X is a subgraph of X' , $\text{sw}(X) < \text{msw}(X) < \text{ksw}(X) < \text{ksw}(X') < \text{ksw}(X)$

Since X is a subgraph of X' , this lemma has an immediate consequence.

Corollary 5. [2]

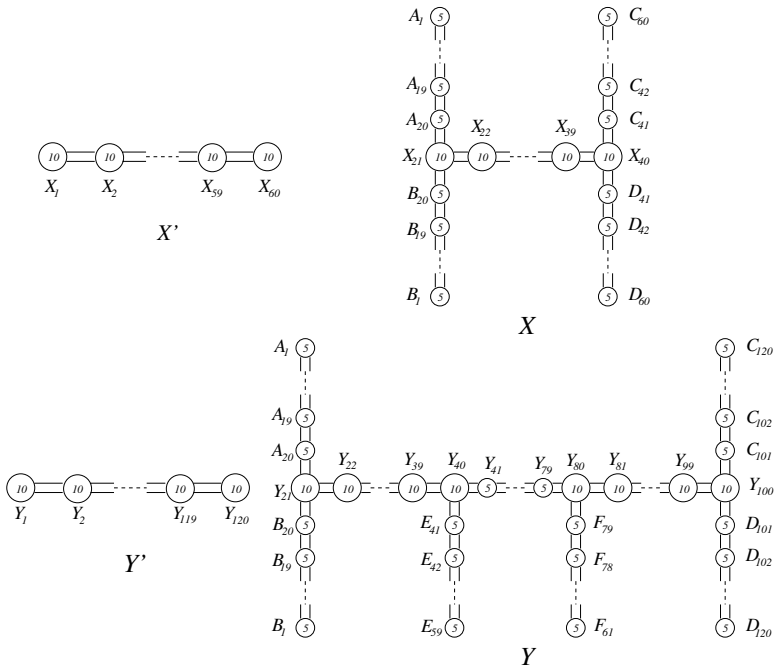


Fig. 2. The graphs X' and Y' and their subgraphs X and Y

In the same vein, let $Y' = K_{10} \square P_{120}$, and Y be as pictured in Figure 2.

Theorem 10. Y is a subgraph of Y' , $\text{msw}(Y) > \text{msw}(Y')$

As before, since Y is a subgraph of Y' , there is an immediate corollary.

Corollary 6. [2]

There are three inequalities in Lemma 1. Corollary 4 shows that the final inequality can be strict and Theorem 9 shows that the first two may also be strict. We can construct a single graph H for which all the inequalities strict.

Theorem 11. For any graph H we have $sw(H) < msw(H) < ksw(H) < mksw(H)$

6.2 Differences Between Sweep Numbers

In Corollary 4 we showed that a graph W existed with $ksw(W) < mksw(W)$. The graph W is very large, containing ≈ 400000 vertices. The large strings of 300 n -cliques were constructed so that sweepers could not “sneak” through, and the C_i and D_i were constructed so that sweepers could “sneak” through.

We showed that the difference between $ksw(W)$ and $mksw(W)$ was 9, though the difference can be much smaller. For instance, we could reduce the size of cliques and length of “paths” by an approximate factor of 5 and then prove that $ksw(W_{\frac{1}{5}}) = 57$ and $mksw(W_{\frac{1}{5}}) = 58$. However, these results, while valid, are more easily demonstrated by using larger cliques and longer paths to make the difference more “believable.”

Similarly, we can construct W_k , where clique size and “path” length are increased by an approximate factor of k . The resulting W_k are “scaled up” versions of W , with appropriately changed sweep numbers. Using arguments similar to those in Theorems 7 and 8, we can prove the following theorem.

Theorem 12. For $k \geq 1$, $ksw(W_k) = 280k + 1$ and $mksw(W_k) = 290k$

In the same manner, we can create families of graphs X_k , X'_k , Y_k , and Y'_k based on X , X' , Y and Y' .

Theorem 13. For $k \geq 1$, $ksw(X_k) = 15k + 1$, $msw(X_k) = 10k + 2$, $ksw(X'_k) = 10k + 1$, $msw(Y) = 15k + 1$ and $msw(Y') = 10k + 1$

Theorem 13 tells us that the difference between the monotonic sweep number and the connected sweep number can be large. As well, it tells us that for connected or monotonic sweeps, a subgraph may need many more sweepers than a supergraph.

Finally, the “Y-square” in Figure 3 is a graph with sweep number 3 and monotonic sweep number 4. (In fact, it is the smallest (edgewise) graph with $sw < msw$.) The “kY-square” is a similarly “scaled up” version of the Y-square. Here, edges are replaced by “paths” of cliques, with each path containing k^2 cliques of size k . This increases the sweep number to $3k + 1$, and the monotonic sweep number to $4k$, again showing that the difference in these values can be quite large.

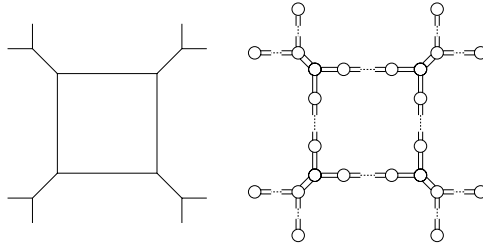


Fig. 3. The Y-square (left) and the kY -square (right)

7 Conclusions

This paper’s main purpose was to demonstrate the increased ease of proof given by constructing graphs with large clique number. This has been done through the construction of the graphs W , X , Y and the kY -square, which allowed us to prove the existence of a graph where the connected sweep number and monotonic connected sweep number differ. Large cliques in a graph not only imply lower bounds on the sweep number, but also allow us to restrict how a graph is cleared by setting up situations where “paths” of cliques must be cleared one at a time rather than by “sneaking” through them.

Having solved one of the open problems in [2], we mention the other: find an upper bound for the ratio $\text{mksw}(G)/\text{sw}(G)$. The bound was discussed in [2] for trees, and the authors believe that it is true for all connected graphs.

As mentioned, the Y-square is the smallest graph with sweep number strictly less than monotonic sweep number. For the other inequalities, we have given examples which show these inequalities can be strict, but these graphs are very large. Smallest graphs with these properties would be interesting to find to examine underlying structures.

Acknowledgements. The authors would like to extend special thanks to Denis Hanson and Xiangwen Li both for their ongoing support and for the fascinating discussion they have provided in our weekly MITACS seminars. The first and third authors was supported in part by NSERC and MITACS.

References

1. B.Alspach, X.Li, and B.Yang, *Searching Graphs and Digraphs*, Manuscript, 2004.
2. L. Barrière, P. Fraignaud, N. Santoro and D. Thilikos, Searching is not Jumping. In Proc. 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003), Lecture Notes in Computer Science, 2880, pp. 34-45, 2003.
3. D. Bienstock and P. Seymour, Monotonicity in graph searching, *Journal of Algorithms* **12** (1991), 239–245.
4. A. S. LaPaugh, Recontamination does not help to search a graph. *Journal of ACM*, 40(1993)224–245.

5. N. Megiddo, S. L. Hakimi, M. Garey, D. Johnson and C. H. Papadimitriou, The complexity of searching a graph. *Journal of ACM* , **35** (1988), pp. 18–44.
6. T. Parsons, Pursuit-evasion in a graph. *Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, pages 426–441, 1976.

A Slightly Improved Sub-cubic Algorithm for the All Pairs Shortest Paths Problem with Real Edge Lengths

Uri Zwick

School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel

Abstract. We present an $O(n^3\sqrt{\log \log n}/\log n)$ time algorithm for the All Pairs Shortest Paths (APSP) problem for directed graphs with real edge lengths. This improves, by a factor of about $\sqrt{\log n}$, previous algorithms for the problem obtained by Fredman, Takaoka and Dobosiewicz.

1 Introduction

The input to the All Pairs Shortest Paths (APSP) problem is a directed graph $G = (V, E)$ with a length function $\ell : E \rightarrow \mathbb{R}$ defined on its edges. The goal is to find, for every pair of vertices $u, v \in V$, the distance from u to v in the graph, and possibly also a shortest path from u to v in the graph. (If there is a path from u to v in the graph that passes through a cycle of negative length, the distance from u to v is defined to be $-\infty$. If there is no path from u to v , the distance is defined to be $+\infty$.)

The APSP problem for directed graphs with real edge weights can be solved in $O(mn + n^2 \log n)$ time by running Dijkstra's [6] Single Source Shortest Path (SSSP) algorithm from each vertex, where $n = |V|$ and $m = |E|$ are the number of vertices and edges, respectively, in the graph. The quoted running time assumes the use of Fibonacci heaps (Fredman and Tarjan [10]), or an equivalent data structure. If some of the edge lengths are negative, then a preprocessing step described by Johnson [15] is necessary. A slightly improved running time of $O(mn + n^2 \log \log n)$ was recently obtained by Pettie [18], based on an approach initiated by Thorup [25], Hagerup [14] and Pettie and Ramachandran [19].

On dense graphs with $m = \Omega(n^2)$, the worst-case running times of the algorithms mentioned above is $\Theta(n^3)$. A running time of $O(n^3)$ is also obtained by the simple Floyd-Warshall algorithm (see [8, 26]). Can the APSP problem be solved in sub-cubic, i.e., $o(n^3)$ time? An affirmative answer was provided by Fredman [9] who showed that the problem can be solved in $O(n^3(\log \log n/\log n)^{1/3})$. This time bound was subsequently improved to $O(n^3\sqrt{\log \log n}/\log n)$ by Takaoka [24], and to $O(n^3/\sqrt{\log n})$ by Dobosiewicz [7]. We present here a further improved algorithm with a running time of $O(n^3\sqrt{\log \log n}/\log n)$.

The complexity of the APSP problem is known to be the same as the complexity of the $n \times n$ matrix multiplication problem (see [1], Theorem 5.7 on page 204). If $A = (a_{ij})$ and $B = (b_{ij})$ are two $n \times n$ matrices, we let $A * B$ be

the $n \times n$ matrix whose (i, j) -th element is $(A*B)_{ij} = \min_k \{a_{ik} + b_{kj}\}$. We refer to $A*B$ as the min-plus product of A and B . (It is trivial to see that the APSP problem can be solved by computing $\log n$ min-plus products. A more intricate argument, given in [1], shows that this extra $\log n$ can be avoided.)

The min-plus product can be naively computed using $O(n^3)$ additions and comparisons. Fredman [9] made the intriguing observation (see also Section 3) that the min-plus product of two $n \times n$ matrices can be inferred after performing only $O(n^{2.5})$ comparisons of sums of two matrix elements! The catch is that Fredman does not specify explicitly which comparisons should be made, nor how to infer the result from the outcome of these comparisons. In more exact terms, Fredman [9] shows that there is a decision tree for computing the min-plus product of two $n \times n$ real matrices whose depth is $O(n^{2.5})$. However, he does not construct such a decision tree explicitly.

Fredman [9] was able, however, to use his observation to obtain an explicit sub-cubic algorithm for the min-plus product, and hence for the APSP problem. This is done by explicitly constructing a decision tree of depth $O(m^{2.5})$ for the min-plus product of two $m \times m$ matrices, where $m = o(\log n)$. The size of this decision tree, which is exponential in m , is $o(n)$. As the product of two $n \times n$ matrices can be solved by computing $(n/m)^3$ products of $m \times m$ matrices, an $o(n^3)$ algorithm is obtained for the problem of multiplying two $n \times n$ matrices.

The main technique used by Fredman [9] to implement his algorithm is Takaoka [24] presents a simpler and more efficient algorithm based on similar ideas. Dobosiewicz [7] uses a somewhat different approach. The speed-up of his algorithm is obtained by using, i.e., the ability to operate simultaneously on $\log n$ bits contained in a single machine word. The exact computational model used by Fredman, Takaoka, Dobosiewicz, and also by us, is discussed in the next section. We stress here that the model is used in all cases, so our improved algorithm is obtained by using a stronger machine model. Our algorithm uses both table look-ups and bit-level parallelism. It uses ideas appearing in the Boolean matrix multiplication algorithm of Arlazarov [3]. It is also inspired by recent dynamic algorithms for the transitive closure in directed graphs (see, e.g., [16, 5, 20]).

Much faster, and truly sub-cubic, algorithms are known for the standard matrix multiplication problem. Strassen [23] obtained an $O(n^{2.81})$ time algorithm. The best available bound is currently $O(n^{2.38})$, obtained by Coppersmith and Winograd [4]. It remains a major open problem whether these techniques could be used to obtain faster algorithms for the min-plus product of matrices with arbitrary edge weights.

Fast algebraic matrix multiplication algorithms were used to obtain faster algorithms for the APSP problem with small integer edge lengths. Zwick [29], improving a result of Alon [2], obtained an $O(n^{2.58})$ algorithm for the APSP problem for unweighted undirected graphs. Even better algorithms are known for undirected graphs with small integer edge weights (see Seidel [21], Galil and Margalit [13, 12], and Shoshan and Zwick [22]). For more results on the APSP problem and its various variants, see Zwick [28].

The rest of this paper is organized as follows. In the next section we discuss the model of computation used. In Section 3 we review the ideas of Fredman [9]. In Section 4 we describe the algorithm of Dobosiewicz [7] on which our algorithm is based. In Section 5 we present the Boolean matrix multiplication algorithm of Arlazarov [3]. Finally, in Section 6 we present our algorithm which uses a combination of ideas from all previous sections. We end in Section 7 with some concluding remarks and open problems.

2 Model of Computation

We use the standard RAM model of computation (see, e.g., [1]). Each memory cell can either hold a number or an w -bit integer. Usually, we assume that $w = \Theta(\log n)$, which is the standard realistic assumption, where n is the input size. All the bounds in the abstract and introduction make this assumption.

Reals numbers are treated in our model as an \dots . The only operations allowed on real numbers are additions and comparisons. (As explained below, subtractions can be simulated in our model.) No conversions between real numbers and integers are allowed. This model is sometimes referred to as the \dots model (see, e.g., [28, 19, 18]).

Although we are mainly interested in the case $w = \Theta(\log n)$, we explicitly describe the dependence of the running times of our algorithm on the word size w . This shows more clearly which logarithmic factors are the result of the word size, i.e., the effect of bit-level parallelism, and which are obtained using other techniques, e.g., table look-up. We always assume that $w \geq \log n$.

The operations allowed on integers are the standard arithmetical and logical operations. We can thus add two integers and compute their bit-wise or. We also assume that we have an instruction that returns the index of one of the 1's in a non-zero word. We do not give an explicit list of the integer instructions assumed. The reason is that when $w = \Theta(\log n)$, which is the case we are most interested in, any conceivable instruction can be emulated, in constant time, using table look-up. In particular, even if there is no instruction for returning the index of, say, the left-most 1 in a non-zero word, we can still find this index, in $O(1)$ time, using table look-up. The time needed for initializing the table will be negligible compared to the other operations performed by our algorithms.

As stated above, our model only allows additions of comparisons of real numbers. It is not difficult to see, however, that allowing subtractions does not change the strength of the model. We simply represent each intermediate result as the difference of two real numbers. When two difference $x_1 - y_1$ and $x_2 - y_2$ need to be compared, we do that by comparing $x_1 + y_2$ and $x_2 + y_1$. It is interesting to note that this simple observation also lies at the heart of Fredman's [9] technique.

Our realistic model of computation should be contrasted with the unrealistic, but nevertheless interesting, model used by Yuval [27]. He shows that the distance product of two matrices with \dots elements can be computed by first converting the elements of the matrix into very high precision real numbers, performing a standard algebraic product, and then converting the elements back

into integers. The conversion steps use the computation of exact exponentials and logarithms. More careful implementations of Yuval’s algorithm, in realistic models of computation, combined with other techniques, form the basis of the algorithms of [13, 12, 22, 29].

3 The Algorithm of Fredman

Let $A = (a_{ij})$ be an $n \times m$ matrix, and let $B = (b_{ij})$ be an $m \times n$ matrix. The distance product $C = A * B$ can be naively computed using $O(mn^2)$ operations. Fredman [9] observed the product can also be computed after performing only $O(m^2n)$ operations.

Theorem 1 (Fredman [9]). Let $A = (a_{ij})$ be an $n \times m$ matrix and $B = (b_{ij})$ be an $m \times n$ matrix. Then the distance product $C = A * B$ can be computed in $O(m^2n)$ operations.

Let $a^i_{rs} = a_{ir} - a_{is}$ and $b^j_{rs} = b_{sj} - b_{si}$, for $i, j \in [n]$ and $s, r \in [m]$. These differences can be formed in $O(m^2n)$ time, and sorted using $O(m^2n \log(mn))$ comparisons. Fredman [9] actually shows that the differences can be sorted using only $O(m^2n)$ comparisons. (For a proof, see Fredman [9].)

Let \bar{a}^i_{rs} be the index of a^i_{rs} in the sorted sequence, and let \bar{b}^j_{rs} be the index of b^j_{rs} in the sequence, for $i, j \in [n]$ and $s, r \in [m]$. While sorting the sequence, we assume that ties are resolved in favor of the a^i_{rs} elements, i.e., if $a^i_{rs} = b^j_{r's'}$, then a^i_{rs} appears before $b^j_{r's'}$ in the sorted sequence and thus $\bar{a}^i_{rs} < \bar{b}^j_{r's'}$. With this convention we have $a^i_{rs} \leq b^j_{r's'}$ if and only if $\bar{a}^i_{rs} \leq \bar{b}^j_{r's'}$.

For every $i, j \in [n]$, we can now find an index $r = r_{ij} \in [m]$ for which $c_{ij} = a_{ir} + b_{rj}$ just by looking at the indices \bar{a}^i_{rs} and \bar{b}^j_{rs} , without looking again at the elements of the matrices A and B . For every $i, j \in [n]$, we want to find an index r for which $a_{ir} + b_{rj} \leq a_{is} + b_{sj}$, for every $s \in [m]$. Note, however, that

$$a_{ir} + b_{rj} \leq a_{is} + b_{sj} \iff a_{ir} - a_{is} \leq b_{sj} - b_{si} \iff a^i_{rs} \leq b^j_{rs} \iff \bar{a}^i_{rs} \leq \bar{b}^j_{rs}.$$

Thus, the outcome of every comparison needed to determine an appropriate index r is implied by the indices computed. □

The above ‘algorithm’ does not explain how to use the indices \bar{a}^i_{rs} and \bar{b}^j_{rs} to determine the indices r_{ij} . It just says that these indices contain enough information to uniquely determine the result.

The fast ‘algorithm’ for rectangular min-plus products of Theorem 1 can be used to obtain a fast ‘algorithm’ for square min-plus products as follows:

Theorem 2 (Fredman [9]). Let $A = (a_{ij})$ and $B = (b_{ij})$ be $n \times n$ matrices. Then the distance product $C = A * B$ can be computed in $O(n^{2.5})$ operations.

Let $1 \leq m \leq n$ be a parameter to be chosen later. We split the matrix A into n/m matrices $A_1, A_2, \dots, A_{n/m}$ of size $n \times m$, and the matrix B into n/m matrices $B_1, B_2, \dots, B_{n/m}$ of size $m \times n$. Clearly, $A * B = \min_{i=1}^k A_i B_i$, where the min here is applied element-wise. Each distant product $A_i B_i$ can be determined, as described in the proof of Theorem 1, using $O(m^2 n)$ comparisons. Computing the n/m products and computing their element-wise minimum thus requires only

$$O\left(\frac{n}{m} \cdot (m^2 n + n^2)\right)$$

comparisons. This expression is minimized for $m = \sqrt{n}$ and the resulting number of comparisons is then $O(n^{2.5})$. \square

We note again that the ‘algorithm’ given in the proof of Theorem 2 is not really an algorithm in the conventional sense of the word, as it does not specify how to infer the result of the distance product from the comparisons performed. More accurately, the theorem says that there is a decision tree for the min-plus product of two $n \times n$ matrices whose depth is $O(n^{2.5})$. Fredman [9] observes, however, that the decision tree whose existence is proved in Theorem 2 can be explicitly constructed for tiny values of n , and this can be used to slightly lower the cost of computing a min-plus product of two $n \times n$ matrices.

Theorem 3 (Fredman [9]). Let $A = (a_{ij})$ and $B = (b_{ij})$ be $n \times n$ matrices. Then the min-plus product $C = A * B$ can be computed using $O(n^3 (\log \log n / \log n)^{1/3})$ comparisons on a machine with $(\log n)$ -bit words.

Takaoka [24] simplified Fredman’s explicit algorithm and reduced its running time to $O(n^3 (\log \log n / \log n)^{1/2})$, again on a machine with $(\log n)$ -bit words.

4 The Algorithm of Dobosiewicz

Dobosiewicz [7] discovered a slightly more efficient explicit implementation of Fredman’s ‘algorithm’ for rectangular min-plus products. Instead of using $(\log n)$ -bit words, as done by Fredman [9] and Takaoka [24], the algorithm of Dobosiewicz simply uses w -bit words.

Theorem 4 (Dobosiewicz [7]). Let A and B be $n \times m$ and $m \times n$ matrices, respectively. Then the min-plus product $C = A * B$ can be computed using $O(m^2 n^2 / w + n^2)$ comparisons on a machine with w -bit words, where $w \leq n / \log n$.

Dobosiewicz’s algorithm is given in Figure 1. It receives an $n \times m$ matrix A , an $m \times n$ matrix B . It returns two $n \times n$ matrices C and R . The matrix C contains the min-plus product $A * B$. The matrix R contains the minimal indices for the product, i.e., $c_{ij} = a_{i,r_{ij}} + b_{r_{ij},j}$, where $r_{ij} \in [m]$, for every $i, j \in [n]$.

The algorithm starts by setting $Z_i \leftarrow [n]$, for $i \in [n]$. The set Z_i contains all the indices j for which c_{ij} and r_{ij} were not determined yet. The algorithm maintains two other collections of sets, X_i , for $i \in [n]$, and Y_s , for $s \in [m]$. The cost of performing operations on these sets will be discussed later. We focus, first, on the correctness of the algorithm.

```

Algorithm  $(C_{n \times n}, R_{n \times n}) \leftarrow MULT(A_{n \times m}, B_{m \times n})$ 

Let  $Z_i \leftarrow \{1, 2, \dots, n\}$ , for  $i \in [n]$ .
for  $r \leftarrow 1$  to  $m$ 
    Let  $a_{rs}^i \leftarrow a_{ir} - a_{is}$ , for  $i \in [n]$ ,  $s \in [m]$ ,  $s \neq r$ .
    Let  $b_{rs}^j \leftarrow b_{sj} - a_{rj}$ , for  $j \in [n]$ ,  $s \in [m]$ ,  $s \neq r$ .
    Form a sorted list  $L$  containing these  $2(m-1)n$  elements.

    Let  $X_i \leftarrow Z_i$ , for  $i \in [n]$ .
    Let  $Y_i \leftarrow \{1, 2, \dots, n\}$ , for  $i \in [n]$ .

    for  $k \leftarrow 1$  to  $2(m-1)n$ 
        if  $L_k$  is  $a_{rs}^i$ , then  $X_i \leftarrow X_i - Y_s$ .
        if  $L_k$  is  $b_{rs}^j$ , then  $Y_s \leftarrow Y_s \cup \{j\}$ .
    end

    for  $i \leftarrow 1$  to  $n$ 
        for every  $j \in X_i$ 
             $r_{ij} \leftarrow r$ .
             $c_{ij} \leftarrow a_{ir} + b_{rj}$ .
        end
         $Z_i \leftarrow Z_i - X_i$ .
    end
end
end
    
```

Fig. 1. The rectangular min-plus multiplication algorithm of Dobosiewicz

The main portion of the algorithm is a loop in which the variable r ranges over the values from 1 to m . In each iteration of the loop the algorithm identifies all pairs of indices (i, j) , where $i, j \in [n]$, for which r is a minimal index, i.e., $(A * B)_{ij} = a_{ir} + b_{rj}$, and for which no other minimal index was found before, and sets the entries c_{ij} and r_{ij} accordingly. (Note that there may be several minimal indices for a pair (i, j) . The algorithm will find the smallest one of them.)

As in the proof of Theorem 1, r is a minimal index for (i, j) if and only if $a_{ir} + b_{rj} \leq a_{is} + b_{sj}$, or equivalently, $a_{ir} - a_{is} \leq b_{sj} - b_{rj}$, for every $s \in [m]$. The algorithm computes the differences $a_{rs}^i = a_{ir} - a_{is}$ and $b_{rs}^j = b_{sj} - b_{rj}$, for every $i, j \in [n]$, $s \in [m]$, and forms a sorted list L containing them. (Again, as in the proof of Theorem 1, if $a_{rs}^i = b_{rs}^j$, then the element a_{rs}^i is placed before b_{rs}^j in the list.) Then, r is a minimal index for (i, j) if and only if a_{rs}^i appears before b_{rs}^j in the list, for every $s \in [m]$.

At the start of each iteration the algorithm sets $X_i \leftarrow Z_i$ for every $i \in [n]$. It then scans the elements of list L , one by one, while maintaining the following invariant: $j \in X_i$ if and only if $j \in Z_i$ and it is ‘still possible’ that r is a minimal index for (i, j) . To be more precise, $j \in X_i$ if and only if $j \in Z_i$ and for every

$s \in [m]$, either a_{rs}^i appears before b_{rs}^j in L , or b_{rs}^j was not scanned yet. To help maintain this invariant, the algorithm also maintains, for every $s \in [m]$, a set Y_s that contains all the indices j for which b_{rs}^j was already encountered.

Let us see what actions should be taken to maintain the invariants when scanning the next element of L . If the scanned element is a_{rs}^i , then all the elements of Y_s should be removed from X_i . (Indeed, if $j \in Y_s$ then b_{rs}^j appears before a_{rs}^i in the list.) The algorithm thus appropriately performs $X_i \leftarrow X_i - Y_s$. If the scanned element is b_{rs}^j , we simply need to add j to Y_s , and the algorithm appropriately performs $Y_s \leftarrow Y_s \cup \{j\}$.

It is easy to see that when all the elements of L are scanned, $j \in X_i$ if and only if $j \in Z_i$ and r is a minimal index for (i, j) . For each $j \in X_i$, the algorithm thus sets $r_{ij} \leftarrow r$ and $c_{ij} \leftarrow a_{ir} + b_{rj}$. The set X_i is then removed from Z_i , for $i \in [n]$. This completes the description and correctness proof of the algorithm.

Let us now discuss the complexity of the algorithm. Consider the cost of a single iteration of the algorithm with a given value of r . Computing the differences a_{rs}^i and b_{rs}^j takes $O(mn)$ time. Sorting them to form the list L takes $O(mn \log(mn))$ time. For each element of L we then perform two set operations on subsets of $[n]$. Each one of the sets X_i, Z_i and Y_s can be represented as an n -bit vector which can be packed into n/w machine words. Each set operation can then be implemented in $O(n/w)$ time. The total cost of an iteration, excluding the cost of the last for loop, is thus $O(mn \log(mn) + mn^2/w)$, which is $O(mn^2/w)$ since we assume that $w \leq n/\log n$. Multiplying this by m , the number of iterations, we get a time bound of $O(m^2n^2/w)$.

To finish the proof it remains to bound the time spent in the double loop in which i ranges from 1 to n and j ranges over all the elements of X_i . To do so, note that for every $i \in [n]$, the m versions of the set X_i obtained in the m iterations are disjoint. This is so because X_i is initialized to Z_i at the beginning of each iteration, and the resulting set X_i is subtracted from Z_i at the end of each iteration. Thus, the total number of elements j extracted from the sets X_i in all iterations is exactly n^2 . Extracting all the elements of a set X_i can be easily done in $O(|X_i| + n/w)$ time using a machine instruction that returns the index of one of the 1's in a non-zero machine word. (See Section 2.) The total time spent in the double loop is therefore at $O(n^2 + mn/w)$. The total running time of the algorithm is therefore $O(m^2n^2/w + n^2)$, as required. \square

Theorem 5 (Dobosiewicz [7]). $n \times n$
 $O(n^3/\sqrt{w})$ $w \leq n/\log n$

As in the proof of Theorem 2, we break the product of two $n \times n$ matrices into n/m products of $n \times m$ by $m \times n$ matrices. The total time needed is then $O(\frac{n}{m} \cdot (\frac{m^2n^2}{w} + n^2))$, which is minimized when we set $m = \sqrt{w}$. \square

Corollary 1 (Dobosiewicz [7]). $n \times n$
 $O(n^3/\sqrt{\log n})$ $(\log n)$

5 The Algorithm of Arlazarov *et al.*

Arlazarov *et al.* [3] considered the different, though related, problem of computing the *or-and* product, i.e., the or-and product, of two Boolean matrices.

Theorem 6 (Arlazarov *et al.* [3]). *Let A be an $n \times \log n$ matrix and B be a $\log n \times n$ matrix. Then the *or-and* product $C = AB$ can be computed in $O(n^2/w)$ time, where w is the word size.*

Let $A = (a_{ij})$ be an $n \times \log n$ matrix, and let $B = (b_{ij})$ be a $\log n \times n$ matrix. Let $C = AB$ be their $n \times n$ Boolean product. We let A_i, B_i and C_i denote i -th row of A, B or C , respectively. As we assume that $w \geq \log n$, each row A_i of A fits into a single machine word. For each row of B and C , on the other hand, requires n/w machine words.

Each row A_i specifies a subset of the rows of B , of size at most $\log n$, that needs to be *or'ed*. Doing this naively would require $O((n \log n)/w)$ time for each row, and thus a total time of $O((n^2 \log n)/w)$.

We can save a $\log n$ factor as follows. For brevity, let $k = \log n$. For every k -bit word $x = x_1 \dots x_k$, we let $BT[x] = \bigvee_{i=1}^k x_i B_i$, i.e., the *or* of the rows of B corresponding to the 1's in the word x . We can compute $BT[x]$, for every k -bit word x , in $O(2^k n/w) = O(n^2/w)$ time. Now $C_i = BT[A_i]$, for every $i \in [n]$. Thus, the rows C_i , for $i \in [n]$, can be looked up in the table B , again in $O(n^2/w)$ time, as required. A complete description of the algorithm is given in Figure 2.

```

Algorithm  $C_{n \times n} \leftarrow BMULT(A_{n \times \log n}, B_{\log n \times n})$ 
 $BT[0] \leftarrow 0$ 
for  $i \leftarrow 0$  to  $\log n - 1$ 
  for  $j \leftarrow 0$  to  $2^i - 1$ 
     $BT[2^i + j] \leftarrow B_i \vee BT[j]$ 
  end
end

for  $i \leftarrow 1$  to  $n$ 
   $C_i \leftarrow BT[A_i]$ 
end
```

Fig. 2. The Boolean matrix multiplication algorithm of Arlazarov *et al*

Theorem 7 (Arlazarov *et al.* [3]). *Let A and B be $n \times n$ matrices. Then the *or-and* product $C = AB$ can be computed in $O(n^3/(w \log n))$ time, where w is the word size.*

As usual, we break the Boolean product of two $n \times n$ matrices into $n/\log n$ products of an $n \times \log n$ matrix by an $\log n \times n$ matrix. The total cost is then $O(\frac{n}{\log n} \cdot \frac{n^2}{w}) = O(\frac{n^3}{w \log n})$. □

The Boolean product of two $n \times n$ matrices can be computed in in $O(n^\omega)$ time, where $\omega < 2.376$ [4] is the algebraic matrix multiplication exponent (see, e.g., Furman [11] and Munro [17]). We do not know, however, how to utilize these fast algebraic or Boolean matrix multiplication algorithms to obtain faster algorithms for the min-plus product of matrices with real elements.

6 The New Algorithm

Using the idea used by Arlazarov [3], we can obtain a slightly more efficient implementation of the algorithm of Dobosiewicz [7].

Theorem 8. *Let X be an $n \times m$ matrix, Y be an $m \times n$ matrix, w be a positive integer, and t be a positive integer. Then the min-plus product $Z = X \otimes Y$ can be computed in $O(m^2 n^2 \log(w \log n) / (w \log n))$ time using w words.*

The improved algorithm is obtained by providing a slightly more efficient implementation of the set operations used by the algorithm of Dobosiewicz [7].

Let t be a parameter to be chosen later. For simplicity, we assume that n/t is an integer. Each dynamic set Y_s , where $s \in [m]$, is maintained as follows. As long as Y_s contains at most t elements, we maintain a simple list \mathcal{Y}_s containing the elements of Y_s . When the size of the list \mathcal{Y}_s reaches t , we prepare a compressed representation Y_s^1 of \mathcal{Y}_s using n/w words, and reset \mathcal{Y}_s to the empty list. Elements added to the set Y_s are again added to the list \mathcal{Y}_s until its length becomes t again. We then prepare a compressed representation of the set $Y_s^2 = \mathcal{Y}_s \cup Y_s^1$ and again empty \mathcal{Y}_s . Continuing in this way, we get $n/t - 1$ snapshots $Y_s^1, \dots, Y_s^{n/t-1}$ of the set Y_s , and the list \mathcal{Y}_s is always of size at most t . For every $s \in [m]$, we let v_s be the index of the last snapshot of Y_s created so far.

A set operation $X_i \leftarrow X_i - Y_s$ is now implemented as follows. We remove the elements in the list \mathcal{Y}_s , one by one, from X_i . This takes only $O(t)$ time. We also set $v_{is} \leftarrow v_s$ to signify that the elements of $Y_s^{v_s}$ should be removed from X_i , but we do not remove these elements as yet. These removals will be carried out at the final stage of the algorithm. Doing all these removals together will allow us to use a trick similar to the one used by Arlazarov [3].

At the end of the sequence of update operations, we need to perform

$$X_i \leftarrow X_i - \bigcup_{s=1}^m Y_s^{v_{is}}, \quad \text{for every } i \in [n].$$

Note that for each $i \in [n]$ and $s \in [m]$ we have $0 \leq v_{is} \leq n/t - 1$. For brevity, let $b = n/t$. It would have been nice to be able to look up the value of $\bigcup_{s=1}^m Y_s^{v_{is}}$ in a table. Unfortunately, such a table will be too large as it would have to contain $b^m = (n/t)^m$ entries, which may be much larger than n . Let $k = \log n / \log(n/t)$. As $b^k = (n/t)^k = n$, we can afford to keep a table with $(n/t)^k$ entries. Thus, we can construct m/k tables, each of size n , such that the g -th table will hold all the sets of the form $\bigcup_{s=gk+1}^{(g+1)k} Y_s^{v_{is}}$. Combining these m/k tables we get a two-dimensional table YT such that

$$YT[g, \sum_{s=gk+1}^{(g+1)k} v_s b^{s-(gk+1)}] = \bigcup_{s=gk+1}^{(g+1)k} Y_s^{v_s}, \quad 0 \leq g < \frac{m}{k}, \quad 0 \leq v_s < b.$$

<u>Initialization:</u>	<u>Finalization:</u>
for $s \leftarrow 1$ to m	$b \leftarrow n/t$
$v_s \leftarrow 0.$	$k \leftarrow \log n / \log b$
$\mathcal{Y}_s \leftarrow \phi.$	for $g \leftarrow 0$ to $m/k - 1$
$Y_s^0 \leftarrow \phi.$	$YT[g, 0] \leftarrow \phi$
end	for $s \leftarrow 1$ to k
	for $v \leftarrow 0$ to $b - 1$
<u>$Y_s \leftarrow Y_s \cup \{i\}:$</u>	for $x \leftarrow 0$ to $b^{s-1} - 1$
if $ \mathcal{Y}_s < t$ then	$YT[g, v \cdot b^s + x] \leftarrow Y_{gk+s}^v \cup YT[g, x]$
$\mathcal{Y}_s \leftarrow \mathcal{Y}_s \cup \{i\}$	end
else	end
$v_s \leftarrow v_s + 1$	end
$Y_s^{v_s} \leftarrow Y_s^{v_s-1} \cup \mathcal{Y}_s$	for $i \leftarrow 1$ to n
$\mathcal{Y}_s \leftarrow \{i\}$	for $g \leftarrow 0$ to $m/k - 1$
end-if	$ind \leftarrow 0$
	for $s \leftarrow gk + 1$ to $(g + 1)k$
<u>$X_i \leftarrow X_i - Y_s:$</u>	$ind \leftarrow b \cdot ind + v_{is}$
$v_{is} \leftarrow v_s$	end
for every $j \in \mathcal{Y}_s$	$X_i \leftarrow X_i - YT[ind]$
$X_i \leftarrow X_i - \{j\}$	end
end	end

Fig. 3. Speeding up the set operations used in the algorithm of Dobosiewicz

Each entry in the table is a subset of $[n]$ represented using n/w machine words. The time needed for constructing the table YT is proportional to its size in words, which is $O((m/k) \cdot n \cdot (n/w)) = O(\frac{mn^2 \log \frac{n}{t}}{w \log n})$. Each set of the form $\cup_{s=1}^m Y_s^{v_{is}}$ can now be formed in $O((m/k) \cdot (n/w))$ time by taking the union of m/k sets found in the table YT . The time needed for computing the n sets $\cup_{s=1}^m Y_s^{v_{is}}$, for $i \in [n]$, is therefore the same as the time needed for preparing the table YT . A full description of the proposed new way of implementing the set operations is given in Figure 3.

Let us now analyze the cost of executing all the set operations performed by the algorithm of Dobosiewicz [7] (see Figure 1) using the new implementation. We bound the cost of the $O(mn)$ set operations performed during one iteration of the algorithm. The total initialization cost is $O(mn/w)$. The total cost of handling the instructions of the form $Y_s \leftarrow Y_s \cup \{i\}$ is $O(m \cdot (n + (n/t)(n/w)))$. The total cost of handling the instructions of the form $X_i \leftarrow X_i - Y_s$ is $O(mn \cdot t)$. Adding the cost of the finalization stage, as discussed above, we get that the total cost of an iteration is

$$O(mnt + \frac{mn^2 \log \frac{n}{t}}{w \log n}),$$

where we have neglected terms that are dominated by the two terms appearing above. To minimize the running time, we choose $t = n/(w \log n)$. The running time of an iteration is then $O\left(\frac{mn^2 \log(w \log n)}{w \log n}\right)$. Multiplying this by the number of iterations we get the time bound claimed. \square

Theorem 9. *Let A and B be $n \times n$ matrices with integer entries. Then the Boolean product $A \cdot B$ can be computed in time $O(n^3/\sqrt{w \log n}/\log(w \log n))$, where w is the word size.*

Yet again, we break the Boolean product of two $n \times n$ matrices into n/m products of an $n \times m$ matrix by an $m \times n$ matrix. We compute each one of these rectangular products using the algorithm of Theorem 8. The total cost is then

$$O\left(\frac{n}{m} \cdot \left(\frac{m^2 n^2 \log(w \log n)}{w \log n} + n^2\right)\right).$$

Choosing $m = \sqrt{w \log n / \log(w \log n)}$, we get the time bound claimed. \square

Corollary 2. *Let A and B be $n \times n$ matrices with integer entries. Then the Boolean product $A \cdot B$ can be computed in time $O(n^3 \sqrt{\log \log n} / \log n)$.*

As an additional corollary, we get the main result of this paper.

Corollary 3. *Let A and B be $n \times n$ matrices with integer entries. Then the Boolean product $A \cdot B$ can be computed in time $O(n^3 \sqrt{\log \log n} / \log n)$.*

7 Concluding Remarks

We have obtained a slightly improved sub-cubic algorithm for the APSP problem with real edge lengths. Unfortunately, we were not able to answer the following major open problem: Is there a genuinely sub-cubic algorithm for the APSP problem with real edge lengths, i.e., an algorithm that runs in $O(n^{3-\epsilon})$ time, for some $\epsilon > 0$?

The algorithm presented here is also the fastest known algorithm for the APSP problem with integer edge lengths taken, say, from the range $\{1, 2, \dots, n\}$. Is there a genuinely sub-cubic algorithm for this version of the problem?

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.
2. N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54:255–262, 1997.
3. V.L. Arlazarov, E.C. Dinic, M.A. Kronrod, and I.A. Faradzev. On economical construction of the transitive closure of a directed graph. *Doklady Akademii Nauk SSSR*, 194:487–488, 1970. English translation in *Soviet Mathematics Doklady*, 11:1209–1210, 1970.
4. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

5. C. Demetrescu and G.F. Italiano. Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In *Proc. of 41st FOCS*, pages 381–389, 2000.
6. E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
7. W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International Journal of Computer Mathematics*, 32:49–60, 1990.
8. R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.
9. M.L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:49–60, 1976.
10. M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
11. M.E. Furman. Application of a method of rapid multiplication of matrices to the problem of finding the transitive closure of a graph. *Doklady Akademii Nauk SSSR*, 194:524, 1970. English translation in *Soviet Mathematics Doklady*, 11:1252, 1970.
12. Z. Galil and O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134:103–139, 1997.
13. Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54:243–254, 1997.
14. T. Hagerup. Improved shortest paths on the word RAM. In *Proc. of 27th ICALP*, pages 61–72, 2000.
15. D.B. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.
16. V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. of 40th FOCS*, pages 81–91, 1999.
17. I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.
18. S. Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004.
19. S. Pettie and V. Ramachandran. Computing shortest paths with comparisons and additions. In *Proc. of 13th SODA*, pages 267–276, 2002.
20. L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. In *Proc. of 43rd FOCS*, pages 679–688, 2002.
21. R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51:400–403, 1995.
22. A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. of 40th FOCS*, pages 605–614, 1999.
23. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
24. T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43:195–199, 1992.
25. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.
26. S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.
27. G. Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Information Processing Letters*, 4:155–156, 1976.
28. U. Zwick. Exact and approximate distances in graphs – a survey. In *Proc. of 9th ESA*, pages 33–48, 2001.
29. U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:289–317, 2002.

Author Index

- Abraham, David J. 3
Ailon, Nir 16
Akutsu, Tatsuya 452
Alspach, Brian 908
Amano, Kazuyuki 28
Amir, Amihood 41
Andersson, Mattias 53
Araki, Toru 442
Armon, Amitai 65
Aronov, Boris 77, 89
Asano, Tetsuo 77, 89
- Bae, Sang Won 101
Bauer, Markus 113
Bazgan, Cristina 124
Beigel, Richard 427
Bengtsson, Fredrik 137
Bereg, Sergey 149, 606
Bilò, Davide 159
Bilò, Vittorio 172
Brandes, Ulrik 184
Brazil, Marcus 196
- Cai, Jin-Yi 209
Calinescu, Gruiã 221, 234
Cechlárová, Katarína 3
Chan, Timothy M. 246
Chan, Wun-Tat 259
Chang, Maw-Shang 282
Chao, Kun-Mao 294
Chazelle, Bernard 16
Chen, Chao 330
Chen, Danny Z. 271
Chen, Hsin-Fu 282
Chen, Jingsen 137
Chen, Kuan-Yu 294
Chen, Xujin 306
Chen, Zhenming 318
Cheng, Ho-Lun 330
Cheng, Qi 342
Chwa, Kyung-Yong 101, 352
Comandur, Seshadhri 16
- Daescu, Ovidiu 669
Dai, H.K. 364
- Dang, Zhe 377
Demaine, Erik D. 1
Demange, Marc 896
de Souza, Criston 778
de Werra, Dominique 896
Dom, Michael 389
Dong, Liang 884
Dragan, Feodor F. 402
Duch, Amalia 415
Dyer, Danny 908
- Escoffier, Bruno 124, 896
- Fan, Hui 884
Flammini, Michele 172
Fu, Bin 427
Fujita, Satoshi 442
Fukagawa, Daiji 452
- Goerdts, Andreas 470
Goldstein, Avraham 484
Goldstein, Darin 496
Golin, Mordecai 508
Gudmundsson, Joachim 53
Guo, Jiong 389
- Haga, Hiroki 693
Hershberger, John 522
Hu, Xiaobo S. 271
Huang, Ming-Deh 342
Hui, Peter 534
Hüffner, Falk 389
- Ibarra, Oscar 377
Iwama, Kazuo 545
- JaJa, Joseph 558, 822
Jansson, Jesper 569, 581
Jaromczyk, Jerzy W. 594
Jiang, Minghui 606
Jo, Byung-Cheol 352
- Katoh, Naoki 77
Kawachi, Akinori 545
Kawada, Taizo 705
Kikuchi, Yosuke 89

- Kim, Hee-Chul 742
 Kim, Jae-Hoon 618
 Klau, Gunnar W. 113
 Knauer, Christian 352
 Kobayashi, Kojiro 496
 Kolman, Petr 484
 Kötter, Rolf 629
 Kowalski, Dariusz R. 644

 Laber, Eduardo Sany 778
 Lanka, André 470
 Lefmann, Hanno 657
 Lerner, Jürgen 184
 Leung, Yiu Cho 508
 Levcopoulos, Christos 53
 Lim, Hyeong-Seok 742
 Liu, Ding 16
 Lomonosov, Irina 402
 Lonc, Zbigniew 594
 Luan, Shuang 271
 Luo, Jun 669

 Mäkinen, Veli 681
 Manlove, David F. 3
 Maruoka, Akira 28
 Mehlhorn, Kurt 3, 77
 Melideo, Giovanna 172
 Miura, Kazuyuki 693
 Moet, Esther 352
 Monnot, Jerome 896
 Mortensen, Christian W. 558
 Moscardelli, Luca 172
 Mount, David M. 2

 Nagamochi, Hiroshi 705
 Nandy, Subhas 89
 Naqvi, Shahid A. 271
 Navarro, Gonzalo 681
 Ngo, Trung Hieu 569
 Niedermeier, Rolf 389
 Nishizeki, Takao 693
 Nor, Igor 41
 Nowakowski, Richard J. 717

 Park, Jung-Heum 742
 Paschos, Vangelis Th. 124, 896
 Pelc, Andrzej 644
 Peng, Zeshan 754
 Pessoa, Artur Alves 766, 778
 Proietti, Guido 159

 Qin, Zhongping 606
 Raitner, Marcus 793

 Sadakane, Kunihiko 681
 Sadjad, Bashir S. 246
 Sasahara, Shinji 89
 Schaefer, Marcus 534
 Schindelhauer, Christian 805
 Shi, Qingmin 558, 822
 Shin Chan-Su 352
 Shparlinski, Igor E. 464
 Shrivastava, Nisheeth 522
 Singh, Vikas 318
 Su, H.C. 364
 Su, Jianwen 377
 Sung, Wing-Kin 569, 581
 Suri, Subhash 522

 Tan, Jinsong 835
 Tan, Xuehou 847
 Ting Hingfung 754
 Tokuyama, Takeshi 77
 Tóth, Csaba D. 522

 Uehara, Ryuhei 859, 871
 Uno, Takeaki 89
 Uno, Yushi 871

 van Oostrum, René 352
 Volbert, Klaus 805
 von Oertzen, Timo 729
 von zur Gathen, Joachim 464

 Wang, Chao 271
 Wang, Lusheng 884
 Wang, Yajun 508
 Wanke, Egon 629
 Watanabe, Osamu 209
 Winter, Pawel 196
 Wong, Prudence W.H. 259

 Xu, Jinhui 318

 Yang, Boting 908
 Yu, Cedric X. 271

 Zachariasen, Martin 196
 Zang, Wenan 306
 Zeh, Norbert 717
 Zelikovsky, Alexander 234

Zhang, Louxin 835
Zheng, Jie 484
Zhu, Binhai 606

Ziegler, Martin 805
Zwick, Uri 65, 921