Fedor V. Fomin
Petteri Kaski (Eds.)

LNCS 7357

Algorithm Theory – SWAT 2012

13th Scandinavian Symposium and Workshops
Helsinki, Finland, July 2012
Proceedings

Springer

Lecture Notes in Computer Science 7357

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Fedor V. Fomin   Petteri Kaski (Eds.)

Algorithm Theory – SWAT 2012

13th Scandinavian Symposium and Workshops
Helsinki, Finland, July 4-6, 2012
Proceedings

Springer

Volume Editors

Fedor V. Fomin
University of Bergen
Institute of Informatics
Postboks 7800, 5020 Bergen, Norway
E-mail: fomin@ii.uib.no

Petteri Kaski
Aalto University
Department of Information and Computer Science
PO Box 15400, 00076 Aalto, Finland
E-mail: petteri.kaski@aalto.fi

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-31154-3 e-ISBN 978-3-642-31155-0
DOI 10.1007/978-3-642-31155-0
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012939846

CR Subject Classification (1998): F.2.2, F.2, G.2.1-2, E.1, G.2, G.1, F.1

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers presented at the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2012), held during July 4–6, 2012, in Helsinki, Finland, co-located with the the 23rd Annual Symposium on Combinatorial Pattern Matching (CPM 2012).

A total of 127 papers were submitted, out of which the Program Committee selected 34 for presentation at the symposium. Each submission was reviewed by at least three members of the Program Committee. In addition, invited lectures were given by Joseph S.B. Mitchell, from State University of New York at Stony Brook and Roger Wattenhofer from ETH Zürich. The program of the co-located CPM 2012 included a further 33 contributed papers and two invited lectures, by Ron Shamir from Tel Aviv University and by Gonzalo Navarro from University of Chile in Santiago.

This year, the Steering Committee decided to initiate the Best Student Paper Award. The Program Committee decided to grant the award to Marek Cygan for the paper titled "Deterministic Parameterized Connected Vertex Cover."

SWAT is held biennially in the Nordic countries; it alternates with the Algorithms and Data Structures Symposium (WADS) and is a forum for researchers in the area of design and analysis of algorithms and data structures. The call for papers invited submissions in all areas of algorithms and data structures, including but not limited to approximation algorithms, parameterized algorithms, computational biology, computational geometry, distributed algorithms, external-memory algorithms, exponential algorithms, graph algorithms, online algorithms, optimization algorithms, randomized algorithms, streaming algorithms, string algorithms, sublinear algorithms, and algorithmic game theory. Starting from the first meeting in 1988, previous SWAT meetings have been held in Halmstad, Bergen, Helsinki, Aarhus, Reykjavík, Stockholm, Bergen, Turku, Humlebæk, Riga, Gothenburg, and Bergen. Proceedings of all the meetings have been published in the LNCS series, as volumes 318, 447, 621, 824, 1097, 1432, 1851, 2368, 3111, 4059, 5124, and 6139.

We would like to thank all the people who contributed to making SWAT 2012 a success. We thank the Steering Committee for selecting Helsinki as the venue for SWAT 2012, and for their help and guidance in different issues. The meeting would not have been possible without the considerable efforts of the local organization teams of SWAT 2012 and CPM 2012. We thank Aalto University, University of Helsinki, and the Federation of Finnish Learned Societies (Tieteellisten seurain valtuuskunta) for their financial and organizational support. The EasyChair conference system provided invaluable assistance in coordinating the submission and review process. Finally, we thank the members of the Program Committee and all of our many colleagues whose timely and meticulous efforts helped the committee to evaluate the large number of submissions and select the papers for presentation at the symposium.

April 2012

Fedor V. Fomin
Petteri Kaski

Conference Organization

Program Committee

Allan Borodin	University of Toronto, Canada
Marek Chrobak	University of California, Riverside, USA
Amin Coja-Oghlan	University of Warwick, UK
Michael Elkin	Ben-Gurion University of the Negev, Israel
David Eppstein	University of California, Irvine, USA
Rolf Fagerberg	University of Southern Denmark
Fedor V. Fomin (Chair)	University of Bergen, Norway
Dimitris Fotakis	National Technical University of Athens, Greece
Fabrizio Grandoni	IDSIA University of Lugano, Switzerland
Michel Habib	LIAFA-Université Paris 7, France
John Iacono	Polytechnic Institute of New York University, USA
Giuseppe Italiano	University of Rome “Tor Vergata”, Italy
Petteri Kaski	Aalto University, Finland
Andrzej Lingas	Lund University, Sweden
Daniel Lokshtanov	University of California, San Diego, USA
Andrzej Pelc	Université du Québec en Outaouais, Canada
Sofya Raskhodnikova	Pennsylvania State University, USA
Saket Saurabh	Institute of Mathematical Sciences, Chennai, India
Christian Sohler	Technische Universität Dortmund, Germany
Kavitha Telikepalli	Tata Institute of Fundamental Research, Mumbai, India
Dimitrios M. Thilikos	National and Kapodistrian University of Athens, Greece
Virginia Vassilevska Williams	University of California, Berkeley, and Stanford, USA
Peter Widmayer	ETH Zürich, Switzerland

Organizing Committee

Matti Järvisalo	University of Helsinki, Finland
Petteri Kaski (Chair)	Aalto University, Finland
Mikko Koivisto	University of Helsinki, Finland
Pekka Orponen	Aalto University, Finland
Valentin Polishchuk	University of Helsinki, Finland
Jukka Suomela	University of Helsinki, Finland
Esko Ukkonen	University of Helsinki, Finland

Steering Committee

Lars Arge	Aarhus University, Denmark
Magnús M. Halldórsson	Reykjavík University, Iceland
Andrzej Lingas	Lund University, Sweden
Jan Arne Telle	University of Bergen, Norway
Esko Ukkonen	University of Helsinki, Finland

External Reviewers

Abu-Khzam, Faisal
Adler, Isolde
Afshani, Peyman
Andoni, Alexandr
Antoniadis, Antonios
Bansal, Nikhil
Barbay, Jérémy
Baswana, Surender
Berman, Piotr
Bhatnagar, Nayantara
Bhattacharya, Binay
Biedl, Therese
Blömer, Johannes
Bohmova, Katerina
Bonsma, Paul
Boucher, Christina
Buchin, Maike
Byrka, Jaroslaw
Cesati, Marco
Chakraborty, Sourav
Chan, T-H. Hubert
Chen, Jianer
Chitnis, Rajesh
Cleary, Sean
Coeurjolly, David
Cormode, Graham
Csirik, Janos
Cygan, Marek
Czyzowicz, Jurek
de Montgolfier, Fabien
Dell, Holger
Disser, Yann
Dobosiewicz, Wlodek
Dragan, Feodor
Ehsanfar, Ebrahim
Elbassioni, Khaled
Englert, Matthias
Erlebach, Thomas
Escoffier, Bruno
Fekete, Sandor
Feldman, Dan
Fellows, Michael
Fernau, Henning
Fischer, Matthias
Flier, Holger
Floderus, Peter
Fort, Marta
Funke, Stefan
Fusco, Emanuele
Gagie, Travis
Ganian, Robert
Gaspers, Serge
Golovach, Petr
Grigorescu, Elena
Guo, Jiong
Göös, Mika
Gørtz, Inge Li
Hajiaghayi, Mohammadtaghi
Halldórsson, Magnús M.
Heggernes, Pinar
Hellweg, Frank
Hermelin, Danny
Hruz, Tomas
Huang, Chien-Chung
Jansen, Bart
Jansen, Klaus
Jha, Madhav
Jiang, Minghui
Johannsen, Daniel
Junttila, Tommi

Kaminski, Marcin
Kanj, Iyad
Kaufmann, Michael
Klein, Rolf
Klimošová, Tereza
Knust, Sigrid
Koivisto, Mikko
Kolliopoulos, Stavros
Komusiewicz, Christian
Korhonen, Janne H.
Korman, Matias
Kortsarz, Guy
Koutris, Paraschos
Kowaluk, Miroslaw
Kreutzer, Stephan
Krivelevich, Michael
Krivosija, Amer
Lampis, Michael
Levcopoulos, Christos
Levin, Asaf
Liu, Yang
Lopez-Ortiz, Alejandro
Lucarelli, Giorgio
Lundell, Eva-Marta
Makris, Christos
Marbach, Peter
McGregor, Andrew
Michail, Dimitrios
Mihalak, Matus
Misra, Neeldhara
Mitra, Pabitra
Mitsche, Dieter
Mnich, Matthias
Molloy, Michael
Montanari, Sandro
Moric, Filip
Mukhopadhyaya, Krishnendu
Muller, Haiko
Munteanu, Alexander
Muthukrishnan, S.
Mäkinen, Veli
Narayanan, Lata
Narayanaswamy, N.S.
Navarro, Gonzalo
Nilsson, Bengt
Nishimura, Naomi
Nussbaum, Yahav
Panagiotou, Konstantinos
Panolan, Fahad
Papamanthou, Charalampos
Patel, Viresh
Paulusma, Daniel
Persson, Mia
Pettie, Seth
Pietrzyk, Peter
Pollefeys, Marc
Popat, Preyas
Poon, Ck
Proskurowski, Andrzej
Pröger, Tobias
Puglisi, Simon
Rafiey, Arash
Raman, Rajiv
Ramanujan, M.S.
Rao, Srinivasa
Ray, Saurabh
Razgon, Igor
Roy, Sasanka
Sabharwal, Yogish
Sach, Benjamin
Saha, Barna
Sanyal, Swagato
Sau, Ignasi
Schacht, Mathias
Schietgat, Leander
Schindelhauer, Christian
Schlotter, Ildikó
Schmidt, Melanie
Schoengens, Marcel
Schulz, André
Schwartz, Roy
Schweitzer, Pascal
Schwiegelshohn, Chris
Segal, Michael
Shachnai, Hadas
Shannigrahi, Saswata
Shapira, Asaf
Sharan, Roded
Siminelakis, Paris
Skopalik, Alexander

Sledneu, Dzmitry
Solis-Oba, Roberto
Solomon, Shay
Souza, Alexander
Speckmann, Bettina
Srivastava, Piyush
Steele, J. Michael
Subramani, K.
Suchy, Ondrej
Tamir, Arie
Tamir, Tami
Tan, Xuehou
Tholey, Torsten
Tiskin, Alexander
Tzamos, Christos
Vahrenhold, Jan
van 't Hof, Pim
van Kreveld, Marc
van Leeuwen, Erik Jan
Verbin, Elad
Vialette, Stéphane
Vidick, Thomas
Villanger, Yngve
Wang, Yusu
Whidden, Chris
Wiese, Andreas
Woeginger, Gerhard
Wolff, Alexander
Woo, Maverick
Yan, Li
Yang, Qifan
Yaroslavtsev, Grigory
Ye, Deshi
Yoshida, Yuichi
Zarges, Christine
Zoros, Dimitris
Zych, Anna

Invited Talks

Visibility Coverage Tours

Joseph S.B. Mitchell
Department of Applied Mathematics and Statistics
State University of New York at Stony Brook
Joseph.Mitchell@stonybrook.edu

The watchman route problem is a classic optimization problem in computational geometry: Determine a shortest path/tour for an observer to be able to view all points within a given geometric domain. The problem combines aspects of the traveling salesperson problem and the set cover problem. While some special cases have polynomial-time exact solutions, most versions of the problem are NP-hard, so attention focuses on approximation algorithms. The problem comes in many varieties, depending on the nature of the domain being searched, assumptions about the searcher(s), and the objective function. We briefly survey the research area of visibility coverage tours and describe some recent advances in approximation algorithms.

Think Global, Act Local

Roger Wattenhofer
Distributed Computing (DISCO)
ETH Zurich, 8092 Zurich, Switzerland
wattenhofer@ethz.ch

The title of my presentation is a motto originally coined in urban design about 100 years ago, but used in various different contexts since. The motto can also be applied to many areas in computer science. For instance, in computational game theory, each agent tries to maximize his/her own (local) benefit, and we analyze the (global) social welfare. Also, computer architecture is designed for locality of reference. Recently, the distributed computing community has made tremendous progress towards understanding the complexity of distributed message passing algorithms. In networks, a rich selection of upper and lower bounds regarding how much time it takes to solve or approximate a problem have been established; we now have a good understanding how local actions and global goal influence each other. This *distributed complexity theory* may ultimately help to give the "think global, act local" motto a mathematical foundation. In my talk I will introduce the message passing model, present a few selected results, mention prominent open problems, and discuss some of the most exciting future research directions. Upcoming applications may not necessarily lie in computer science but rather in other areas that deal with networked systems, e.g. biology or social sciences.

Table of Contents

α-Visibility[⋆]

Mohammad Ghodsi[1,3,⋆⋆], Anil Maheshwari[2], Mostafa Nouri[1],
Jörg-Rüdiger Sack[2], and Hamid Zarrabi-Zadeh[1]

[1] Department of Computer Engineering,
Sharif University of Technology, Tehran, Iran
{ghodsi,zarrabi}@sharif.edu, nourybay@ce.sharif.edu
[2] School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada
{anil,sack}@scs.carleton.ca
[3] Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

Abstract. We study a new class of visibility problems based on the notion of *α-visibility*. Given an angle α and a collection of line segments $\mathcal{S}$ in the plane, a segment t is said to be α-visible from a point p, if there exists an empty triangle with one vertex at p and the side opposite to p on t such that the angle at p is α. In this model of visibility, we study the classical variants of point visibility, weak and complete segment visibility, and the construction of the visibility graph. We also investigate the natural query versions of these problems, when α is either fixed or specified at query time.

1 Introduction

The study of visibility is at least 99 years old, when in 1913 Brunn [5] proved a theorem about the kernel of a set. By now, visibility has become one of the most studied notions in computational geometry. The reasons are two-fold: 1) such problems arise naturally in areas where computational geometry tools and algorithms find applications. 2) their solutions are required, or serve as building blocks in the development of solutions to other problems, such as motion planning problems. Many natural problem instances arise and have been extensively studied in two and higher dimensions. The reader is referred to [3,8].

Previous Work: Given a polygonal scene $\mathcal{S}$, the *visibility polygon* of a point p, denoted by $\mathrm{VP}(p)$, is the set of all points inside the scene that are visible from p. When the scene is a simple polygon or a polygonal domain, several algorithms exist to compute the visibility polygon of a point with/without preprocessing. Previous results for point-visibility inside a scene are summarized in Table 1.

Given a segment s, the *weak visibility polygon* $\mathrm{VP}(s)$ of s is the set of points in the scene that are visible from at least one point on s. Guibas *et al.* [11] showed how to compute the weak visibility polygon of a segment inside a simple polygon in $O(n)$ time. Suri and O'Rourke [22] established that the weak visibility polygon of a segment inside a polygon with holes has size $\Theta(n^4)$, but can be represented

[⋆] Research supported by NSERC, HPCVL and SUN Microsystems.

[⋆⋆] This author's research was partially supported by IPM under grant No: CS1389-2-01.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 1–12, 2012.

Table 1. Summary of the previous work on point-visibility. Here, n is the total complexity of the scene, h is the number of holes, m_p is the complexity of VP(p) for a query point p, and $|E|$ is the number of edges in the visibility graph of the scene.

Scene	Prep. Time	Space	Query Time	Ref.
polygon	—	$O(n)$	$O(n)$	[3],[7],[15]
polygon	$O(n^3 \log n)$	$O(n^3)$	$O(\log n + m_p)$	[4]
polygon	$O(n^3)$	$O(n^3)$	$O(\log n + m_p)$	[12]
polygon	$O(n^2 \log n)$	$O(n^2)$	$O(\log^2 n + m_p)$	[1]
polygonal domain	—	$O(n)$	$O(n \log n)$	[22]
polygonal domain	—	$O(n)$	$O(n + h \log h)$	[13]
polygonal domain	$O(n^2)$	$O(n^2)$	$O(n)$	[2]
polygonal domain	$O(n^2 \log n)$	$O(n^2)$	$O(m_p \log(n/m_p))$	[23]
polygonal domain	$O(n^3 \log n)$	$O(n^3)$	$O(\min\{h, m_p\} \log n + m_p)$	[24]
convex polygons	$O(n \log n + \|E\|)$	$O(\|E\|)$	$O(m_p \log n)$	[20]

by a set of $O(n^2)$ triangles. They also gave an algorithm for computing the weak visibility polygon of a segment inside a polygon with holes in $O(n^4)$ time.

The *visibility graph* of a polygon is the undirected graph of the visibility relation on the vertices of the polygon. Optimal algorithms for computing visibility graphs exist. Ghosh and Mount [9] established its construction in $O(n \log n + |E|)$ time for a polygon with holes. Here, $|E|$ is the number of edges in the resulting visibility graph.

The *weak visibility graph* of a set of segments is defined as the graph, with a node for each segment and an edge between any pair of weak visible segments, that have at least two mutual visible points. Ghosh and Mount [9] and Keil *et al.* [14] computed the weak visibility graph in $O(n \log n + |E|)$ time. Nouri *et al.* [19] demonstrated how to detect the visibility relation between two query segments in $O(n^{1+\varepsilon})$ time, using $O(n^2)$ space and $O(n^{2+\varepsilon})$ preprocessing time, for any fixed ε. Gudmundsson and Morin [10] obtained similar results for testing weak visibility relation between a query point and a segment.

New Model: Let $\mathcal{S}$ be a set of n line segments in the plane, which are non-intersecting except possibly at their end-points. Since each polygonal scene is composed of a set of segments, $\mathcal{S}$ can model polygonal scenes as well. Let α be a positive real number. In this paper, we study a new class of visibility problems based on the notion of *α-visibility* as follows:

Point-visibility: A segment $t \in \mathcal{S}$ is said to be α-visible from a point p, if p can see t with an angle at least α; that is, if there exists an empty triangle with one vertex at p and side opposite to p on t such that the angle at p equals α (Fig. 1.a).
Segment-visibility: A segment t is said to be *weakly α-visible* from a segment s, if there is a point on s from which t is α-visible (Fig. 1.b). A segment t is said to be *completely α-visible* from s, if for all points on s, t is α-visible (Fig. 1.c).
Visibility Graph: We define the *weak* (respectively *complete*) *α-visibility graph* of $\mathcal{S}$ as a directed graph G_α whose vertices are the segments of $\mathcal{S}$, and for any two

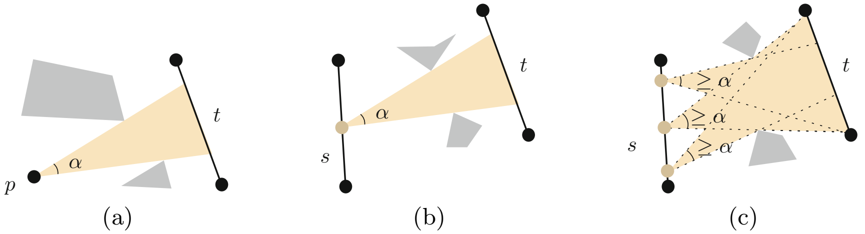


Fig. 1. (a) Segment t is α-visible from p. (b) Segment t is weakly α-visible from segment s. (c) Segment t is completely α-visible from segment s.

segments $s, t \in \mathcal{S}$, there is a directed edge from s to t if t is weakly (respectively, completely) α-visible from s.

The notion of *α-visibility* appears to be natural. Typically, all optical/digital imaging devices have limitations, quantified by their resolutions. Our α-visibility model is capable of capturing this limitation, and provides a more realistic alternative to the classical visibility models studied in the literature. The value α could be also employed to approximate the inaccuracy of a device used to provide visibility-related measurements. E.g., laser rangefinders do not return any data when they are too far off in angle from the surface normal. In general, there is a wealth of literature on approximation algorithms for geometric shortest path problems, and there is a close connection between visibility and shortest path problems, but still there are essentially no results on the notion of approximate visibility. This paper lays down a foundation in that respect, and will likely inspire further study of the notion of approximate visibility.

Our Results: In this paper, we present the first results for several variants of the point/segment visibility problems in the α-visibility model. The main idea that we use is to group the set of all possible visibility directions into $O(1/\alpha)$ directions, provide insights into α-visibility, and then employ a combination of geometric tools, such as trapezoid diagrams, shortest path maps, ray shooting, range searching, etc., to solve the visibility problem in each group of directions. In the following, $\mathcal{S}$ denotes the scene, i.e., the set of input segments in the plane.

In Section 3, we present efficient data structures that enable answering queries of the form "Is segment $t \in \mathcal{S}$ α-visible from a query point p in the plane?". When α is fixed, we preprocess $\mathcal{S}$ in $O(n \log n)$ time into a data structure of size $O(n)$ that answers aforementioned point-visibility queries in $O(\log n)$ time[1]. We also provide data structures for answering point-visibility queries when α is specified at the query time.

In Section 4, we show that the (weak and complete) α-visibility graph of $\mathcal{S}$ has linear size (in n), and that it can be computed in $O(n \log n)$ time. Once the graph has been computed in $O(n \log n)$ time, queries of the form "Is $t \in \mathcal{S}$ weakly/completely α-visible from $s \in \mathcal{S}$?" can be answered in $O(1)$ time. Then,

[1] The running times and space bounds of the data structures presented in this paper involve a factor $1/\alpha$, which is omitted when α is assumed to be a fixed constant.

we show how to preprocess $\mathcal{S}$ in $O(n \log n)$ time into a data structure of size $O(n)$ such that queries of the form "Is segment $t \in \mathcal{S}$ weakly α-visible from a query segment s in the plane?" can be answered in $O(\log n)$ time.

Note that one of the key differences between standard visibility (i.e., when $\alpha = 0$) and α-visibility lies in the size of the weak/complete visibility graph of the line segments in the plane. While the former has quadratic size, the latter is linear in size, which makes it appealing both theoretically and from an applied perspective especially when dealing with large data sets.

2 Preliminaries

Ray Shooting: A typical range shooting problem in the plane has the following form: given a set of n segments in the plane, build a data structure that, for any query ray r, report the first segment intersected by r quickly.

Theorem 1 (Chan [6]). *Given a set of n line segments in the plane, there is a data structure requiring $O(n \log^3 n)$ preprocessing time and $O(n \log^2 n)$ space, such that we can find the first point of intersection between a query ray and the set in $O(\sqrt{n} \log^2 n)$ expected time.*

Splinegons (or informally curved polygons) are defined as generalizations of polygons [21]. A splinegon S is formed from a polygon P by replacing one or more edges of P with curved edges such that the region bounded by each curved edge and the segment joining its end-points is convex.

Theorem 2 (Melissaratos and Souvaine [17]). *Given a simple splinegon S with n edges, there is a data structure requiring $O(n)$ preprocessing time and $O(n)$ space, such that, for any query point p and a ray $\mathbf{r}$ emanating from p, the first intersection of $\mathbf{r}$ with the splinegon can be reported in $O(\log n)$ time.*

Simplified Trapezoidal Diagram: Given a set $\mathcal{S}$ of segments in the plane and a direction d, we define a subdivision of the plane such that, each region in the subdivision is the maximal region with the property that all points in that region see the same segment in direction d. We can construct this subdivision by drawing a line in the reverse direction of d, from each end-point of all the segments of $\mathcal{S}$, until it meets another segment. We call this subdivision, the *simplified trapezoidal diagram* of $\mathcal{S}$ in direction d, and denote it by $\mathcal{T}_d(\mathcal{S})$.

Theorem 3. *Given a set $\mathcal{S}$ of n segments in the plane and a direction d, we can construct the simplified trapezoidal diagram $\mathcal{T}_d(\mathcal{S})$ by a plane-sweep in the direction perpendicular to d in $O(n \log n)$ time and $O(n)$ space.*

Shortest Path Maps: For a point s in a simple polygon P, the *shortest path map*, $SPM(s)$, is a partition of P into cells such that for all points t in a cell, the sequence of vertices of P along the shortest path from s to t is fixed. It is well-known that the complexity of $SPM(s)$ is $O(n)$ and it can be built in $O(n)$ time, where n is the number of vertices in P. If we preprocess $SPM(s)$ for point location, for a query point p we can find in $O(\log n)$ time, the last vertex of P in the shortest path from s to p, which can be thought of as the root associated to the cell containing p.

3 Point Visibility

In this section, we show how to build data structures that efficiently determine, for a query point p in the plane and a query segment $t \in \mathcal{S}$, whether t is α-visible from p. First, let $\alpha > 0$ be a fixed constant.

Visibility testing for fixed α:

Theorem 4. *We can preprocess $\mathcal{S}$ into a data structure of size $O(n)$ in $O(n \log n)$ time, such that α-visibility testing can be carried out in $O(\log n)$ time.*

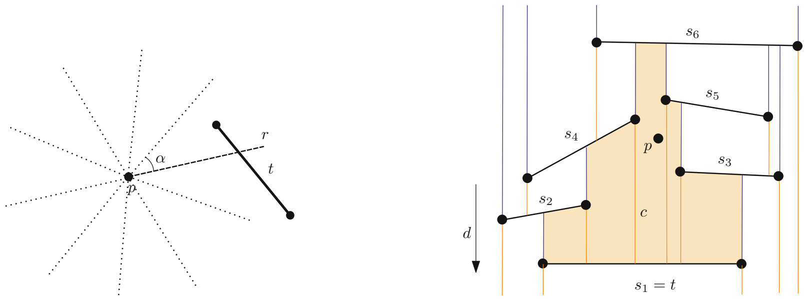


Fig. 2. (a) Segment t is α-visible from p, and r intersects t. (b) A trapezoidation of a set of segments in direction d.

Proof. Assume that we have a set of $\lceil \frac{2\pi}{\alpha} \rceil$ rays emanating from p, as in Fig. 2a, so that the angle between any two consecutive rays is α (except possibly between a pair, where it is $\leqslant \alpha$). Let D denote the set of directions of these rays. If t is visible from p with an angle at least α, t must intersect one of the rays drawn from p as in Fig. 2a. Let r be a ray that intersects t and let d be the direction of r. Consider the trapezoidal diagram of $\mathcal{S}$ in direction d, as in Fig. 2b. Observe that p lies inside the trapezoid that sees t in direction d. Therefore, if t is α-visible from p, p must be inside a trapezoid that sees t, in the trapezoidal diagram drawn for $\mathcal{S}$, based on directions in D.

It remains only to be checked whether p sees t with an angle of at least α. Consider the simplified trapezoidal diagram $\mathcal{T}_d(\mathcal{S})$ in direction d. Note that p is inside a region c, whose visible segment in direction d is t (see Fig. 2b)[2]. For a point p in c, the shortest paths from p to the end-points of t, inside c, consist of two convex chains. The maximum visible part of t from p, including the intersection point of r with t, is determined by extending the first edge of each of the shortest paths. Therefore, to compute the maximum visible part of t from p, it is sufficient to find the first turning points on the shortest paths from p to the end-points of t, in c.

[2] Note that the region c is essentially a simple polygon.

The complexity of the trapezoidal map in direction d is $O(n)$ and can be computed in $O(n \log n)$ time. We can use the trapezoidal map to locate the trapezoid containing p and find the visible segment in direction d. The shortest path map of the end-points of each segment t, in the corresponding cell c, has complexity proportional to the size of c. Since, the sum of the complexities of cells in the simplified trapezoidal map is $O(n)$, all the shortest path maps can be computed in $O(n \log n)$ time using $O(n)$ space. We repeat this construction for all directions of D. To answer a query, for each direction d, we locate the trapezoid in which p lies, in the trapezoidal map of $\mathcal{S}$. This can be done in $O(\log n)$ time. The trapezoid gives us a segment that is visible in direction d. If the segment is not t we proceed to the next direction. Otherwise, based on the shortest path map associated to the cell of the simplified trapezoidal map, we can find the maximum portion of t that is visible from p, and check if that portion forms an angle of at least α with t. If so, we report "yes", otherwise we check the next direction. The first turning point in the shortest path from p to the end-points of t can be located in $O(\log n)$ time, and since the number of directions is $O(1/\alpha)$ (a constant), the total query time is $O(\log n)$. □

Visibility Testing for Non-fixed, Constant α: Our objective here is to build a data structure, such that given a query point p, and a query segment $t \in \mathcal{S}$ and an angle $\alpha > 0$, we need to determine if t is α-visible from p.

Theorem 5. *We can preprocess $\mathcal{S}$, in $O(n \log^3 n)$ time, into a data structure of size $O(n \log^2 n)$, such that we are able to detect α-visibility of query segment $t \in \mathcal{S}$ from a query point p in $O(\sqrt{n} \log^3 n)$ expected time.*

Proof. Assume that we have a set of $\lceil \frac{2\pi}{\alpha} \rceil$ rays emanating from p, so that the angle between any two consecutive rays is at most α (Fig. 2a). If t is α-visible from p, then it is visible from p along at least one of these rays. Let r be such a ray. To check if the visible part around the intersection point of r with t constitutes an angle $\geqslant \alpha$, we need to find the maximum visible part of t from p around that intersection point. If the visible part forms an angle $\geqslant \alpha$, we answer to the visibility query affirmatively. Therefore, we need to solve the following two sub-problems: *i*) What is visible from p along r? *ii*) If we rotate r around p, when does the visibility from p along r change?

Problem (i) is ray shooting among segments, which has already been discussed in Theorem 1. We use range searching to solve problem (ii) as follows. Preprocess the end-points of segments in $\mathcal{S}$, for the two level half-plane range searching problem, and for each canonical subset of the result, we compute the convex hull of its points. Without loss of generality, assume that we want to find the first visibility change when we rotate r counterclockwise. The result has $O(\sqrt{n} \log^2 n)$ canonical sets, and for each canonical set we have pre-computed the convex hull of its points. Hence, for each canonical set, we can find the first point visited from that set, while we rotate r counterclockwise around p. The final result is the point that is visited first among all such points. Therefore, it can be found in $O(\sqrt{n} \log^3 n)$ expected time. The total complexity is now derived from the complexities of the two sub-problems. □

Remark: Using techniques introduced by [16], one can achieve a space/query-time tradeoff for the problem. Using $O(m)$ space, for any $n \log^2 n \leqslant m \leqslant n^2$, one obtains a query time of $O((n/\sqrt{m}) \operatorname{polylog} m)$.

4 Segment Visibility

The weak α-visibility graph, G_α, for $\mathcal{S}$ is defined as follows. Each segment of $\mathcal{S}$ is associated to a unique vertex in G_α. Furthermore, for any two segments $s, t \in \mathcal{S}$, if t is weakly α-visible from s, then there is a directed edge in G_α from the vertex corresponding to s to the vertex corresponding to t.

Lemma 1. *The weak α-visibility graph G_α of $\mathcal{S}$ has linear size.*

Proof. Fix a set of $O(1/\alpha)$ directions, D, such that the angle between any two adjacent directions is at most α. Assume that $t \in \mathcal{S}$ is weakly α-visible from $s \in \mathcal{S}$. Then, there is a point $p \in s$ that sees t with an angle at least α. Observe that there exists a direction $d \in D$ that is inside the angle of view from p. Let r be the segment connecting p to t in direction d. Clearly, r does not intersect any other segment of $\mathcal{S}$. Now slide r in the direction perpendicular to d, while it still connects s to t, until r meets an end-point of a segment in $\mathcal{S}$. One of two cases may arise: (i) r reaches an end-point of s or t, or (ii) r reaches an end-point of a segment other than s or t. In the first case, an end-point of s sees t in direction d, or an end-point of t sees s in the direction opposite to d. In the second case, a segment exists in $\mathcal{S}$ for which one of its end points sees t in direction d. Hence, segment-to-segment weak α-visibility can be mapped to a *unique* point-to-segment α-visibility. The number of directions is constant, and the number of end-points of segments in $\mathcal{S}$ is $O(n)$. The total number of distinct segment pairs (s, t), such that t is weakly α-visible from s, is $O(n)$. □

Theorem 6. *$\mathcal{S}$ can be preprocessed into a data structure of size $O(n)$ in $O(n \log n)$ time, so that weak α-visibility testing for any two query segments $s, t \in \mathcal{S}$ can be carried out in $O(1)$ time.*

Proof. Firstly, we compute G_α. Recall the set-up in the proof of Lemma 1. Assume that $t \in \mathcal{S}$ is weakly α-visible from $s \in \mathcal{S}$ and $p \in s$ sees t with an angle of at least α. Let d be the direction in D, that is inside the angle of view of p. Assume that the segment r connects p to t in direction d. Now slide r without changing its direction until it meets an end-point of a segment. The region so swept is a strip w,in which every point on s, on one side of w, sees a point on t, on the opposite side of w, in direction d. Thus, w is a trapezoid in the trapezoidal map of $\mathcal{S}$, in direction d. We denote the nodes associated with s and t in the weak α-visibility graph by s^* and t^*, respectively. Therefore, the first condition that must be met for an edge from s^* to t^* is that s and t are two facing (i.e., opposite) edges of a trapezoid in one of the trapezoidal maps constructed for directions in D.

While this condition is necessary, it is not sufficient. To check whether t is actually weakly α-visible from s, we need to find a point on s that can see t with

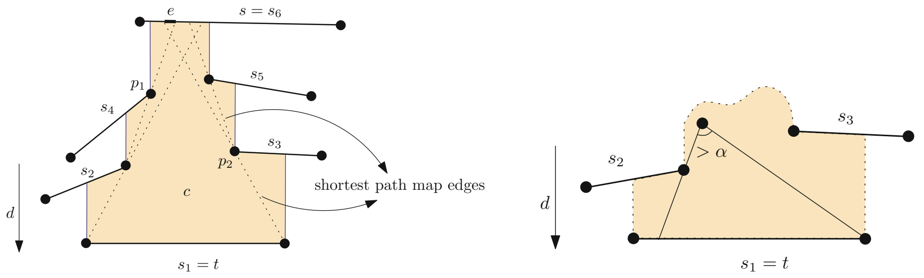


Fig. 3. (a) Segment s is partitioned into sub-segments. (b) The α-visibility region in direction d for $t \in \mathcal{S}$ is shaded.

an angle $\geqslant \alpha$. We do so by partitioning s into sub-segments, such that for each sub-segment, the shortest paths are combinatorially identical (i.e., the shortest paths to the end-points of t have the same set of turning points). This can be done by computing the shortest path map of the endpoints of t inside its adjacent region in the simplified trapezoidal map, as in the previous section, and finding the intersection points of s with the edges of the shortest path maps. Assume that s is partitioned into such subsegments. Let e be one of the subsegments (see Fig. 3a). We now want to determine the point on e which has the greatest view angle to t. Since all points on e have the same combinatorial shortest paths, the largest angle of view from each point on e towards t is determined by the two fixed points. These are the first turning points, say p_1 and p_2, in the shortest paths from any point in e to the end-points of t. The problem thus reduces to finding a point on e with the maximum view through p_1 and p_2. This point is on the intersection of the smallest circle through p_1 and p_2 that intersects e. Therefore, if the circle through p_1 and p_2 which is tangent to the supporting line of e, is incident on the segment e itself, then that supporting point on e will have the largest view. Otherwise, it will be one of the end-points of e. If the view angle is $\geqslant \alpha$, then add the edge from s^* to t^* in G_α. The above procedure is repeated for each sub-segment of s.

The complexity of shortest path map is $O(n)$ and each edge in the map can intersect at most one segment from $\mathcal{S}$. Therefore, the total number of sub-segments is $O(n)$. For each sub-segment, the first turning points on the shortest paths to the end-points of a visible segment can be found in $O(\log n)$ time. Given the turning points, the point with the greatest angle of view can be determined in $O(1)$ time. Therefore, G_α can be computed in $O(n \log n)$ time. For each direction $d \in D$, let $G_d = (V, E_d)$ be the subgraph of G_α with only edges $(s^*, t^*) \in E$ such that s and t are two facing edges in $\mathcal{T}_d$. Obviously G_d is planar and can thus be stored using $O(n)$ space. Checking whether a pair of vertices is connected by an edge takes constant time [18]. In order to determine if t is weakly α-visible from s, we need to verify the existence of edge (s^*, t^*) in G_d for each $d \in D$. This requires $O(1/\alpha)$ time, which is a constant. □

4.1 One Arbitrary Query Segment

Next, we want to build a data structure, such that given two query segments $s \notin \mathcal{S}$ and $t \in \mathcal{S}$, we can determine if t is weakly α-visible from s. As part of the preprocessing, we compute, for each segment $t \in \mathcal{S}$, the set of all points in the plane from which t is α-visible. If t is weakly α-visible from s, then s must have a point in this set. First, we define the *α-visibility region in direction d for t*, as the region containing the points from which t is α-visible and d is inside the angle of view (see, Fig. 3b).

Lemma 2. *For any direction d and the segment t, the boundary of the α-visibility region, say r_t, in direction d for t consists of two monotone curves with respect to the direction perpendicular to d.*

Corollary 1. *The α-visibility region for any $t \in \mathcal{S}$ and for any direction d, does not contain a hole.*

Lemma 3. *The total complexity of the α-visibility regions, in direction d, for all the segments in $\mathcal{S}$, is linear.*

Proof. Let c denote the cell in $\mathcal{T}_d$ associated to t, and let r_t denote the α-visibility region in direction d for t. Obviously, $r_t \subseteq c$. Construct shortest path maps from the end-points of t inside c. Our first claim is that the boundary of r_t intersects each edge of the shortest path map at most once. By contradiction, as shown in Fig. 4a, assume that for an end-point of t, say t_0, there is an edge e in the shortest path map of t_0, such that the boundary of r_t intersects e at least twice. We can choose two points $p, q \in e$, such that $p \in r_t$ and $q \notin r_t$ and $\overrightarrow{pq}$ points towards t. Let p_1 and p_2 be the two extreme points on t visible from p through its angle of view. Let p_1 be closer to t_0 than p_2. Because p and q are on an edge of the shortest path map of t_0, both are visible from p_1. Moreover, p_1 is defined by the intersection of t and the supporting line of e. We know that $\angle p_1pp_2 \geqslant \alpha$. Now observe that $\angle p_1qp_2$ is greater than $\angle p_1pp_2$ and is empty. Therefore, q can see t with angle $\geqslant \alpha$, and because $q \in c$, it is also in r_t, which contradicts the assumption that $q \notin r_t$. Therefore, the boundary of r_t intersects each edge of the shortest path map at most once.

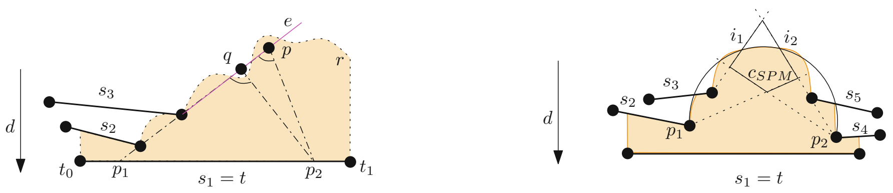


Fig. 4. (a) Points p and q are two points on e. p is in the α-visibility region in direction d for segment t while q is outside the region. (b) Points i_1 and i_2 are two consecutive points which are the intersections of the α-visibility region, in direction d for the segment t, with c_{SPM}.

Consider next the example shown in Fig. 4b. Let i_1 and i_2 be two consecutive intersection points of the boundary of r_t with the two shortest path maps of the end-points of t. Then, i_1 and i_2 are two points on the boundary of a cell, c_{SPM}, in the overlay of the two shortest path maps. Our second claim is that the boundary of r_t between i_1 and i_2 has complexity proportional to the size of c_{SPM}. This holds because there is a combinatorially unique shortest path in c_{SPM} towards the end-points of t. Therefore, for all points on the boundary of r_t, between i_1 and i_2, the largest view to t is determined by two fixed points, say p_1 and p_2. The set of all points which can see t through p_1 and p_2, with an angle $\geqslant \alpha$, are inside, or on the boundary of, the circle through p_1, p_2 having inscribed angle α lying on p_1p_2 clipped by segment p_1p_2. Hence, the boundary of r_t between i_1 and i_2 is determined by the intersection of that circle with c_{SPM}. This intersection has complexity at most $2 * |c_{SPM}|$ because any edge in c_{SPM} can be intersected by the circle at most twice. The complexity of r_t is equal to the number of intersections of its boundary with the shortest path maps and segments of $\mathcal{S}$. Thus, the size of r_t is $O(|c|)$. The total complexity of the cells in $\mathcal{T}_d$ is linear. Therefore, the total complexity of the α-visibility regions in direction d, for all the segments, is $O(n)$. □

Lemma 4. *The α-visibility region in direction d, for all segments in $\mathcal{S}$, can be computed in $O(n \log n)$ time using $O(n)$ space.*

Proof. Let r_t denote the α-visibility region in direction d for t. We first compute $\mathcal{T}_d$ for $\mathcal{S}$. For each segment $t \in \mathcal{S}$, we construct the two shortest path maps of the end-points of t in the cell c associated to t in $\mathcal{T}_d$. Let t_0 and t_1 be the end-points of t; let $SPM(t_0)$ and $SPM(t_1)$ denote the shortest path maps of the end-points, respectively. By definition, the end-points of t are in r_t. We compute the boundary of r_t, by traversing the boundary of c from t_0 until we reach t_1. The events are the intersection points of r_t with $SPM(t_0)$ and $SPM(t_1)$ and the boundary of c. In the first part of r_t, the shortest paths to t_0 and t_1 are two direct segments. The set of points with this property that can see t with angle at least α lies inside or on the boundary of a circle through t_0 and t_1 and on one side of t. We need to compute the intersections of this circle with c, $SPM(t_0)$ and $SPM(t_1)$. The first intersection point with $SPM(t_0)$ and $SPM(t_1)$ affects one of the shortest paths. For this shortest path, the last turning point is changed. Therefore, the circle needs to be updated so that it passes through this point and the last turning point of the other shortest path. After this update, we continue with updated paths in a similar manner until we reach t_1.

$\mathcal{T}_d$ can be computed in $O(n \log n)$ time. $SPM(t_0)$ and $SPM(t_1)$ can be computed in $O(n)$ time. Computing the first intersection point of the current circle with $SPM(t_0)$ and $SPM(t_1)$ takes time proportional to the complexity of the current cells in $SPM(t_0)$ and $SPM(t_1)$. This yields a total time of $O(|c|)$ for all circles. Computing the intersection points of each circle with c can also be done in $O(|c|)$ total time as well. Therefore, the α-visibility regions in direction d for all segments can be computed in $O(n \log n)$ time using $O(n)$ space. □

Theorem 7. *We can preprocess $\mathcal{S}$ into a data structure of size $O(n)$ in $O(n \log n)$ time, such that given two query segments $s \notin S, t \in \mathcal{S}$, their weak α-visibility can be tested in $O(\log n)$ time.*

Proof. As before, we first fix a set D of $O(1/\alpha)$ directions with the property that the angle between any two adjacent directions is at most α. If t is weakly α-visible from s, then there is a direction d in D, and a point q on s from which t is α-visible, and q can see t in direction d inside its angle of view. It is easy to see that q is in the α-visibility region in direction d for t. So the problem reduces to checking the intersection of s with each of the α-visibility regions computed for t, with respect to all directions in D.

We compute the α-visibility regions in all directions $d \in D$ for all segments $t \in \mathcal{S}$ and preprocess each region for ray shooting queries. An α-visibility region is a bounded, hole-free region and its boundary consists of straight line segments and circular arcs. Therefore, it is a splinegon and we can use Theorem 2 for ray shooting queries. Given two query segments $s \notin \mathcal{S}$ and $t \in \mathcal{S}$, we first find r_t (the α-visibility region in direction d for t). We need to know if s has any point in r_t. Let s_0 and s_1 be the end-points of s. We first check whether $s_0 \in r_t$, but because the ray shooting algorithm [17] has point location as a basis, we can use it and determine if $s_0 \in r_t$. If this is the case, t is α-visible from s_0 and weakly α-visible from s. If $s_0 \notin r_t$, shoot a ray in r_t (the splinegon) to find the first intersection point of the ray originating from s_0 in the direction towards s_1. If the intersection point is on s itself, then s intersects r_t and therefore t is weakly α-visible from s, otherwise it is not.

Computing the α-visibility regions for $\mathcal{S}$ takes $O(n \log n)$ time and $O(n)$ space. Preprocessing the α-visibility regions for ray shooting queries takes the same time and space. Point location and ray shooting queries can be performed in $O(\log n)$ time. This estalishes the bound claimed in the theorem. □

4.2 Complete α-Visibility

We want to build a data structure such that given two query segments $s, t \in \mathcal{S}$, we can determine if t is completely α-visible from s. A segment t is *completely α-visible from another segment s* if and only if t is α-visible from all points on s. Note that the complete α-visibility graph is a subgraph of the weak α-visibility graph, and hence its size is linear. We follow a scheme similar to that in Theorem 6. Here, we need to ensure that the collection of all the points on s that can see t, with respect to α-visibility, cover whole of s. For this, we employ shortest path maps of end-points of t. The details are similar to that in Theorem 6.

Theorem 8. *We can preprocess $\mathcal{S}$ into a data structure of size $O(n)$ in $O(n \log n)$ time, such that we can answer complete α-visibility testing queries in $O(1)$ time.*

References

1. Aronov, B., Guibas, L., Teichmann, M., Zhang, L.: Visibility queries and maintenance in simple polygons. Discrete Comput. Geom. 27(4), 461–483 (2002)
2. Asano, T., Asano, T., Guibas, L., Hershberger, J., Imai, H.: Visibility of disjoint polygons. Algorithmica 1(1), 49–63 (1986)

3. Asano, T., Ghosh, S., Shermer, T.: Visibility in Plane in the Handbook in Computational Geometry. Elsevier Science (1999)
4. Bose, P., Lubiw, A., Munro, J.: Efficient visibility queries in simple polygons. In: Proc. 4th Canadian Conf. Comput. Geom., pp. 23–28 (1992)
5. Brunn, H.: Über Kerneigebiete. Math. Ann. 73, 436–440 (1913)
6. Chan, T.M.: Optimal partition trees. In: Proceedings of the 2010 Annual Symposium on Computational Geometry, SoCG 2010, pp. 1–10. ACM, N.Y. (2010)
7. ElGindy, H.A., Avis, D.: A linear algorithm for computing the visibility polygon from a point. J. Algorithms 2(2), 186–197 (1981)
8. Ghosh, S.: Visibility Algorithms in the Plane. Cambridge University Press (2007)
9. Ghosh, S., Mount, D.: An output-sensitive algorithm for computing visibility. SIAM J. Comput. 20, 888–910 (1991)
10. Gudmundsson, J., Morin, P.: Planar visibility: testing and counting. In: Proceedings of the 2010 Annual Symposium on Computational Geometry. ACM (2010)
11. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2, 209–233 (1987)
12. Guibas, L., Motwani, R., Raghavan, P.: The robot localization problem. SIAM J. Comput. 26(4), 1120–1138 (1997)
13. Heffernan, P., Mitchell, J.: An optimal algorithm for computing visibility in the plane. SIAM J. Comput. 24(1), 184–201 (1995)
14. Keil, M., Mount, D., Wismath, S.: Visibility stabs and depth-first spiralling on line segments in output sensitive time. Int. J. Comput. Geometry Appl. 10(5), 535–552 (2000)
15. Lee, D.: Visibility of a simple polygon. Computer Vision, Graphics, and Image Processing 22(2), 207–221 (1983)
16. Matoušek, J.: Range searching with efficient hierarchical cuttings. Discrete & Computational Geometry 10, 157–182 (1993)
17. Melissaratos, E., Souvaine, D.: Shortest paths help solve geometric optimization problems in planar regions. SIAM Jl. Computing 21(4), 601–638 (1992)
18. Munro, J., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. 31, 762–776 (2002)
19. Nouri, M., Zarei, A., Ghodsi, M.: Weak Visibility of Two Objects in Planar Polygonal Scenes. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007, Part I. LNCS, vol. 4705, pp. 68–81. Springer, Heidelberg (2007)
20. Pocchiola, M., Vegter, G.: The visibility complex. Int. J. Comput. Geometry Appl. 6(3), 279–308 (1996)
21. Souvaine, D.: Computational geometry in a curved world (algorithm). PhD thesis. Princeton University, Princeton (1986), AAI8629439
22. Suri, S., O'Rourke, J.: Worst-case optimal algorithms for constructing visibility polygons with holes. In: Proceedings of the 1986 Annual Symposium on Computational Geometry, pp. 14–23 (1986)
23. Vegter, G.: The Visibility Diagram: A Data Structure for Visibility Problems and Motion Planning. In: Gilbert, J.R., Karlsson, R. (eds.) SWAT 1990. LNCS, vol. 447, pp. 97–110. Springer, Heidelberg (1990)
24. Zarei, A., Ghodsi, M.: Query point visibility computation in polygons with holes. Comput. Geom. Theory Appl. 39, 78–90 (2008)

Partial Matching between Surfaces Using Fréchet Distance[⋆]

Jessica Sherette and Carola Wenk

Department of Computer Science, University of Texas at San Antonio, USA
{jsherett,carola}@cs.utsa.edu

Abstract. Computing the Fréchet distance for surfaces is a surprisingly hard problem. We introduce a *partial* variant of the Fréchet distance problem, which for given surfaces P and Q asks to compute a surface $R \subseteq Q$ with minimum Fréchet distance to P. Like the Fréchet distance, the partial Fréchet distance is NP-hard to compute between terrains and also between polygons with holes. We restrict P, Q, and R to be coplanar simple polygons. For this restricted class of surfaces, we develop a polynomial time algorithm to compute the partial Fréchet distance and show that such an $R \subseteq Q$ can be computed in polynomial time as well. This is the first algorithm to address a partial Fréchet distance problem for surfaces and extends Buchin et al.'s algorithm for computing the Fréchet distance between simple polygons.

Keywords: Computational Geometry, Shape Matching, Fréchet Distance.

1 Introduction

The *Fréchet distance* is a similarity metric for continuous shapes such as curves and surfaces. It is defined via continuously mapping points between the shapes while minimizing the maximum distance between mapped points. For two curves, a popular illustration is to consider a man and a dog continuously walking on one of the curves each, while being connected by a leash. They can vary their relative speeds but cannot move backwards. The distance of the curves is the minimum leash length required for both to walk along these curves. For surfaces, one can intuitively consider continuously morphing one surface to the other. The "leash" required by a particular morphing is the maximum distance from any point in one surface to its morphed image in the other surface. The Fréchet distance is the smallest leash required by any possible morphing between the surfaces.

While the Fréchet distance between polygonal curves can be computed in polynomial time [2], computing it between surfaces is much harder. It has been shown that it is NP-hard to compute the Fréchet distance between a triangle and a self-intersecting surface [9], as well as between terrains and between polygons

[⋆] This work has been supported by the National Science Foundation grant NSF CAREER CCF-0643597.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 13–23, 2012.

with holes [3]. And while in [1] it was shown to be upper semi-computable, it remains an open question whether the Fréchet distance between general surfaces is even computable. On the other hand, in [5] a polynomial time algorithm was given for computing the Fréchet distance between two simple polygons. This was the first paper to give any algorithm for computing the Fréchet distance for a nontrivial class of surfaces and remains the only known approach. This approach has recently been generalized to folded polygons [6].

While the Fréchet distance compares the entirety of two surfaces, it is natural to also consider matching all of one surface to some part of the other one. This yields a *partial* variant of the Fréchet distance. The notion of partial Fréchet distance between curves was introduced in [2]. For surfaces, however, partial Fréchet distance has not been considered to date. We introduce the following *partial* Fréchet distance computation problem for simple coplanar polygons. See Section 2 for a formal definition of partial Fréchet distance for surfaces.

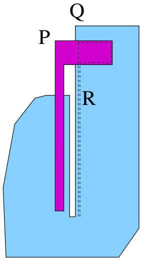


Fig. 1. Partial Fréchet distance example

Our definition of partial Fréchet distance is an extension to surfaces of the one introduced for curves in [2]. Our algorithm is based on the algorithm for simple polygons [5]. We first identify a set of polygons in Q which are similar to P but possibly not homeomorphic to it, see Section 3. We next prove that we can always generate a polygon in this set with Fréchet distance at most ε to P, and we give a polynomial time algorithm to find such a polygon, see Section 4.

2 Preliminaries

Let P and Q be two simple coplanar polygons with m and n vertices, respectively. The Fréchet distance is defined as

$$\delta_F(P,Q) \;=\; \inf_{\sigma:\, P\to Q}\ \sup_{p\in P}\ \|p-\sigma(p)\|$$

where $\|\cdot\|$ is the Euclidean norm and σ ranges over orientation-preserving homeomorphisms that map each point $p\in P$ to an image point $q=\sigma(p)\in Q$. We assume that a triangulation for P is given. We refer to the interior triangulation edges of P as *diagonals*. We denote the boundary of a simple polygon R by ∂R. In this work we give a polynomial time algorithm to solve the following partial Fréchet decision variant.

Definition 1. (Partial Fréchet distance) *Given two coplanar triangulated simple polygons P and Q and some $\varepsilon > 0$, decide whether there exists a simple polygon $R\subseteq Q$ such that $\delta_F(P,R)\leq\varepsilon$.*

Deciding if a polygon $R\subseteq Q$ exists which P can be mapped to within Fréchet distance ε requires considering many mappings of the points in P. A natural approach to decide whether such a mapping exists is to use the free space diagram. This data structure was introduced to decide if two polygonal curves are within Fréchet distance ε [2].

For curves $f, g\colon [0,1] \to \mathbb{R}^d$ and $\varepsilon \geq 0$, *free space* is defined as $FS_\varepsilon(f,g) = \{(s,t) \in [0,1]^2 \mid ||f(s) - g(t)|| \leq \varepsilon\}$. The partition of $[0,1]^2$ into free space and non-free space is the *free space diagram*. A monotone path, starting at the bottom left corner of the free space diagram and traveling entirely through free space to the top right corner, exists if and only if $\delta_F(f,g) \leq \varepsilon$.

3 Computing a Valid Set of Neighborhoods

In this section we present an algorithm based on the one by Buchin et al. to compute the Fréchet distance between simple polygons [5]. Specifically, given two simple coplanar polygons P and Q and some ε, our algorithm computes what we refer to as a valid set of neighborhoods which encodes many candidate polygons in Q to map P to. In Section 4 we prove that among these many, possibly self-intersecting, polygons there is a simple one which is within Fréchet distance ε to P.

3.1 Valid Set of Neighborhoods

The simple polygons algorithm uses the free space diagram to map ∂P to ∂Q [5]. All possible mappings can thus be checked with a 2D free space diagram of the boundary curves, while performing additional checks to ensure that the diagonals of P are mapped correctly. In our case, ∂P can be mapped to the interior of Q as well. Therefore, a simple extension to the previous algorithm would be to use a 3D free space diagram $[0,1] \times Q$, in which the x-axis corresponds to ∂P and the (y,z)-axes correspond to the entire polygon Q, see Figure 2 for an example. With this modification, the free space diagram once again can account for all possible mappings of the vertices in ∂P. Let $D_\varepsilon(p)$ represent the closed L_2-disk of radius ε centered at p.

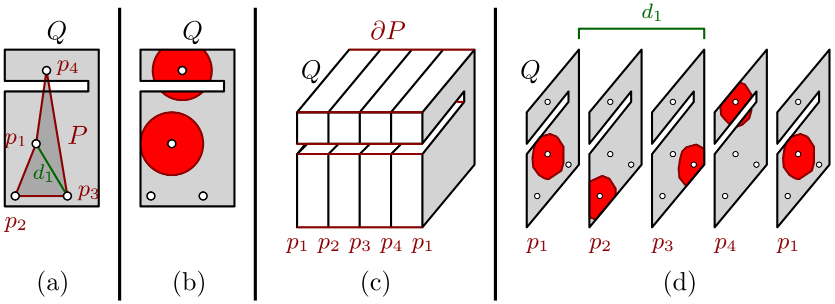


Fig. 2. (a) Example simple polygons P and Q. (b) There are two neighborhoods around the vertex p_4 but only one around p_1. (c) Example 3D free space diagram between ∂P and Q. (d) Associated with each vertex along ∂P are several neighborhoods in Q.

Definition 2. (Neighborhood) *For a vertex $p \in P$ its image point $p' \in Q$ must be in $D_\varepsilon(p) \cap Q$. We define a* neighborhood *of a vertex $p \in P$ as a maximal connected subset of $D_\varepsilon(p) \cap Q$, see Figure 2(b).*

Let $\mathrm{SP}_Q(a, b)$ denote the shortest path in Q between a and b.

Definition 3. ((Q, ε)-valid set of neighborhoods) *Let $S = \{p_1, p_2, \ldots, p_m\}$ be a sequence of vertices along the boundary of a simple polygon P. Let $N = \{N_1, N_2, \ldots, N_m\}$ be a set of neighborhoods where N_i is the chosen neighborhood for vertex p_i. Every sequence of vertices $S' = \{p'_1, p'_2, \ldots, p'_m\}$ with $p'_i \in N_i$ defines a mapping from S to S' by mapping each $p_i \in S$ to $p'_i \in S'$.*

The set of neighborhoods N is (Q, ε)-valid *if and only if every S' defines a mapping such that for all line segments $\overline{p_i p_j}$ in the triangulation of P the $\delta_F(\overline{p_i p_j}, SP_Q(p'_i p'_j)) \leq \varepsilon$.*

While this definition uses all combinations of points in the set of neighborhoods the following lemma states that it suffices to check only one set of image points S'. It follows from the same argument used by Buchin et al. [5] to prove Lemma 11 for diagonals and hourglasses. Instead of having hourglasses between curves we now have hourglasses between two-dimensional connected sets of points. Due to lack of space the proof of this is deferred to the full version of the paper.

Lemma 1. *The set of neighborhoods N is (Q, ε)-valid if and only if there exists some S' which defines a mapping such that for all line segments $\overline{p_i p_j}$ in the triangulation of P the $\delta_F(\overline{p_i p_j}, SP_Q(p'_i p'_j)) \leq \varepsilon$.*

Given a (Q, ε)-valid set of neighborhoods for a simple polygon P, consider each triangle T in the triangulation of P. The vertices of T are mapped to points in Q by some S' associated with the (Q, ε)-valid set of neighborhoods. Connecting each pair of mapped vertices with shortest paths in Q yields an image polygon T' in Q. By the definition of a (Q, ε)-valid set of neighborhoods it must be the case that $\delta_F(T, T') \leq \varepsilon$. Ideally, the union of all such T' would yield a polygon $R \subseteq Q$ such that $\delta_F(P, R) \leq \varepsilon$. Unfortunately, a pair of image polygons T'_1 and T'_2 in Q may overlap. In such a case the mapping between P and R is not a homeomorphism and the Fréchet distance of R and P would not be defined. In Section 4 we show, given a (Q, ε)-valid set of neighborhoods, how to find a polygon $R' \subseteq Q$ such that $\delta_F(P, R') \leq \varepsilon$.

3.2 Algorithm and Runtime Analysis

In this section we prove the following theorem.

Theorem 1. *Given some $\varepsilon \geq 0$ and two simple coplanar polygons P and Q we can compute a (Q, ε)-valid set of neighborhoods in P if such a set exists in $O(m^3 n)$ time.*

Proof. For all $p \in P$ we compute $FS_\varepsilon(p, Q)$. We refer to this as a *slice* of $FS_\varepsilon(P, Q)$. A slice consists of all of the neighborhoods of a vertex. Since Q

contains n vertices, each vertex $p \in P$ has at most $O(n)$ neighborhoods associated with it. We can compute $Q \cap D_\varepsilon(p)$ in time $O(n)$ for a total of $O(mn)$ for all of the $O(m)$ slices.

Next, we test whether there exists a path in the free space diagram which is monotone along ∂P and which starts and ends at the same point in Q. We do this by repeatedly computing the reachability between pairs of slices of the free space diagram. This yields a graph which encodes the reachability information for the entire free space diagram, specifically, the reachability between the neighborhoods of the slices. Note that the path through the free space diagram does not have to be monotone along Q, so reaching any point in a neighborhood is the same as reaching all points. Thus, a neighborhood is reachable from another neighborhood if and only if some pair of points in them are connected by a shortest path with Fréchet distance ε to the original line segment in P. We can decide if the Fréchet distance between a line segment and polygonal curve is at most ε in $O(n)$ time. To compute the reachability between a pair of slices we need to do this for all $O(n^2)$ pairs of their neighborhoods. We refer to this as *merging* slices. In each step we only need to consider the two consecutive slices being merged, see Figure 3(a),(b). It takes $O(m)$ merges to propagate the reachability information across the entire free space diagram. Thus the run time to find a monotone path through the free space diagram is $O(n^3m)$.

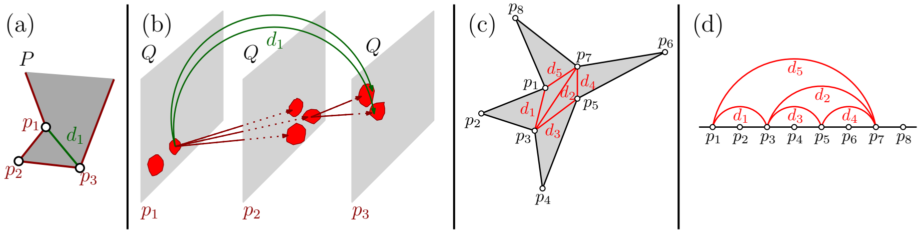


Fig. 3. (a) Example portion of a simple polygon P. (b) We examine the slices associated with each vertex and propagate reachability between the neighborhoods and perform additional checks for the diagonals. (c) Example simple polygon P. (d) This is the nesting structure of the diagonals of P where ∂P has been cut open along the line segment $\overline{p_8p_1}$.

A monotone path through the free space diagram is not sufficient to find a valid set of neighborhoods since we must also check that the diagonals of P are within Fréchet distance ε to their respective shortest path image curves. This adds some additional dependency between the slices of the free space diagram which we must account for, see Figure 3(b). We account for this in the same way as Buchin et al. by carefully choosing which order to merge the slices. Specifically, consider cutting the polygon ∂P open at a point, see Figure 3(c). This yields a nesting structure for the diagonals of P, see Figure 3(d). Instead of simply merging the slices left to right along ∂P we merge them from the bottom to the top of the nesting. This does not significantly increase the number of merges or their complexity. Thus the run time is $O(n^3m)$. □

4 Computing a Simple Polygon

In this section we prove that P has a (Q, ε)-valid set of neighborhoods if and only if there exists a simple polygon $R \subseteq Q$ with $\delta_F(P, R) \leq \varepsilon$. In particular, we prove the following theorem.

Theorem 2. *There exists a simple polygon R such that $\delta_F(P, R) \leq \varepsilon$ if and only if P has a (Q, ε)-valid set of neighborhoods. We can compute such a simple polygon R in time $O(m^2n)$.*

It is trivial to see that if there exists a simple polygon $R \subseteq Q$ such that $\delta_F(P, R) \leq \varepsilon$ then P and Q must have a (Q, ε)-valid set of neighborhoods. By definition, the points in such a simple polygon R would be associated with a (Q, ε)-valid set of neighborhoods. However, the other direction is non-trivial to prove and is stated and proven as Theorem 4 in Section 4.2. Interestingly, proving this requires solving a variant of the *constrained embedding problem*, the general version of which was shown to be NP-hard [8,9]. Due to lack of space, description of our variant as well as the original problem are omitted.

As seen in Section 3.2, we can check whether two simple polygons, P and Q, have a (Q, ε)-valid set of neighborhoods in time $O(m^2n)$. To compute the partial Fréchet distance of P and Q we need to find the minimum ε such that a (Q, ε)-valid set of neighborhoods exists. Similar to [5], we can find a set of critical values for ε and then perform a binary search on them to find the optimal value of ε. Due to lack of space, discussion of the critical values is deferred to the full version of the paper. This yields the following theorem.

Theorem 3. *The partial Fréchet distance can be computed between simple coplanar polygons P with m vertices and Q with n vertices in time $O(mn^3 \log(mn))$ where P is mapped to Q.*

4.1 Theorem 4 and Related Lemmas

Let P_k be a triangulated planar graph of k vertices in P where a pair of vertices, a and b, are connected in P_k if and only if the line segment $\overline{ab}$ is in the triangulation of P. Let polygon(P_k) be the simple polygon defined by the graph P_k. Let $\sigma\colon$ polygon$(P_k) \to Q$ be a mapping and $\sigma[\text{polygon}(P_k)]$ be the resulting image of polygon(P_k) in Q. In this section we prove the following theorem which yields the missing direction of Theorem 2 with $R = \sigma[\text{polygon}(P_m)]$.

Theorem 4. *Let P and Q be simple coplanar polygons with m and n vertices respectively and for which there exists a (Q, ε)-valid set of neighborhoods. There exists a homeomorphism $\sigma\colon$ polygon$(P_k) \to Q$ such that $\delta_F($polygon$(P_k), \sigma[$polygon$(P_k)]) \leq \varepsilon$. We can compute such a mapping in time $O(k^2n)$.*

To prove this theorem we use an inductive proof on k. This yields a polynomial time algorithm to construct $\sigma[\text{polygon}(P_m)]$ iteratively. We map each of the vertices of P within the neighborhood associated with it in the given (Q, ε)-valid set of neighborhoods. Thus, at each iteration we map a vertex a in P to a point a' in the neighborhood N_a. The mapped image, $\sigma[\text{polygon}(P_k)]$, of polygon$(P_k)$ influences this choice of a' since it partially covers N_a. Specifically, the point a' must be chosen in $N_a \cap \overline{\sigma[\text{polygon}(P_k)]}$. We refer to a maximal connected subset of $N_a \cap \overline{\sigma[\text{polygon}(P_k)]}$ as *region*. A neighborhood is divided into multiple regions and we must choose one region to map a vertex in P to.

Unfortunately, in some cases there are multiple, viable, choices for which region to map a. The choice of region cannot be made arbitrarily since an image polygon may be added later which completely covers the chosen region. Naturally, it is computationally expensive to just consider all combinations of regions for all neighborhoods. Fortunately, we can identify a unique region of a neighborhood which we call the *original region* and prove the following lemma about it. The intuitive idea of the original region is that $D_\varepsilon(a)$ is divided by *spikes* of ∂P into multiple preimage regions. The original preimage region is the one which contains a, see Figure 4(a). The original region is the one which is bounded by the images of the spikes which bounded the original preimage region, see Figure 4(b). By mapping each point to an image point in its associated original region we maintain the relative order of points in P in R. Due to lack of space the proof of this lemma and formal definition of the original region are deferred to the full version of the paper.

Lemma 2. *Given some P_k and image $\sigma[\text{polygon}(P_k)]$. Let R be the original region of a vertex $a \in P$. The set $R \setminus \sigma[\text{polygon}(P_k)]$ is non-empty.*

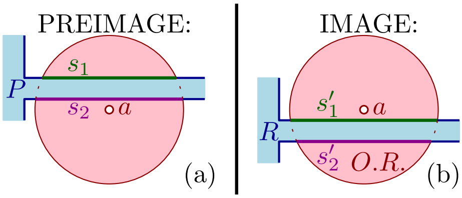


Fig. 4. (a) The spike s_2 bounds the original preimage region. (b) Likewise the spike s'_2 bounds the original region.

In addition, the following lemma restricts the cases which we must consider in proving Theorem 4. Due to lack of space the proof of this lemma is omitted.

Lemma 3. (Crossing Lemma) *Let Q be a simple polygon and let $\overline{a_1 b_1}$ and $\overline{a_2 b_2}$ be non-crossing line segments embedded in same plane as Q. Let f'_1 and f'_2 be shortest paths in Q such that $\delta_F(\overline{a_1 b_1}, f'_1) \leq \varepsilon$ and $\delta_F(\overline{a_2 b_2}, f'_2) \leq \varepsilon$. We refer to $D_\varepsilon(a_1) \cup D_\varepsilon(b_1)$ as the* endpoint portion *of $\overline{a_1 b_1}$.*

If f'_1 and f'_2 cross, then either f_1 intersects the endpoint portion of $\overline{a_2 b_2}$ or f_2 intersects the endpoint portion of $\overline{a_1 b_1}$.

4.2 Proof of Theorem 4

The proof proceeds by induction on k. For the base case, choose three vertices which form a triangle T in P. Add these vertices to P_k. Connect the image points of the vertices of T in Q by shortest paths in Q. Because they are shortest paths it must be the case that none of them cross and the image T' of T is a simple polygon. By the definition of a valid set of neighborhoods, the Fréchet distance between T and the new image polygon T' is less than or equal to ε.

For the inductive step, we assume the theorem statement is true for k. We then show it holds for $k+1$ by adding a vertex $a \in P$ to P_k which forms a triangle in P with two vertices in P_k. Other than this, the choice of a is arbitrary. We must choose some vertex a' in Q to map a to. For every vertex $b \in P_k$ where a and b are adjacent in P_k, we must add an image curve from a' to b'. It may be that some these added image curves cross the existing image curves in σ[polygon(P_k)]. Let d be a line segment whose image curve d' crosses image curve from a' to b'. From Lemma 3, we know one of two cases has occurred. Either the endpoint portion of $\overline{ab}$ was crossed by d' or the endpoint portion of d was crossed by the image curve from a' to b'. Because both d' and b' were present in σ[polygon(P_k)] which is assumed to be a simple polygon we do not need to consider the case that $D_\varepsilon(b)$ is crossed by d'. In the following two sections we outline a method for avoiding such crossings. We first describe how to place the point a'. This yields an image curve from a' to b'. We next show how to update image curves such as d' where the endpoint portion of its preimage d are crosses by the image curve from a' to b'. By avoiding these two cases we ensure that no crossings occur. Note that we preprocess Q in time $O(n)$ to allow for shortest path queries in $\log(n)$ time [10,11].

Placement of the New Image Point. We now define how this new image point a' of a should be placed. As discussed in Section 4.1, we map a to a point in the original region. From Lemma 2 we know that a can be mapped to a point a' in Q which is not already in σ[polygon(P_k)]. In the worst case σ[polygon(P_k)] contains $O(k)$ shortest paths each with complexity $O(n)$ for a total complexity of $O(kn)$. Intersecting $D_\varepsilon(a)$ with each line segment in each shortest path in σ[polygon(P_k)] takes time $O(kn)$. Choose a' in the portion of $D_\varepsilon(a)$ not covered by σ[polygon(P_k)]. We then compute the shortest path in Q from a' to each point b' where b' is the image of some vertex $b \in P_k$ and a and b are adjacent in P_k. In the worst case we compute $O(k)$ shortest paths each of complexity $O(n)$ for a total of $O(kn)$ time.

Modification of Old Image Points. A new image curve, f, may yet cross the neighborhoods of other vertices in P. The image points of these vertices need to be updated if their associated image curves cross the newly added ones. We map each of these to an image point on the part of f which crosses their neighborhood. Therefore the new image point will also be in the original neighborhood associated with its preimage by the (Q,ε)-valid set of neighborhoods.

Moving these points changes their shortest paths and thus their associated image curves. This in turn could cause additional image points to need to be updated if they were crossed by the these moved image curves. To avoid this cascading effect we only update the portion of the old image curve that crosses f into T' while the rest of it remains in place. The point that the old image curve crossed f serves as a meeting point for these two paths. For an example see Figure 5. A new image curve f is added and the image curve d_1' for $\overline{ac} = d_1$ must be collapsed to f. b is the vertex on d which is mapped to the intersection of d_1' and f. The vertex a is now mapped to a point a'' on f. The line segment $\overline{ab}$ is now mapped to the shortest path between a'' and b'. The line segment $\overline{bc}$ is mapped to the shortest path between b' and c'. This effectively splits the line segment d_1 into two distinct parts. Fortunately, we can show that each original line segment in P can be split into at most three pieces. Due to lack of space the proof of this is deferred to the full version of the paper.

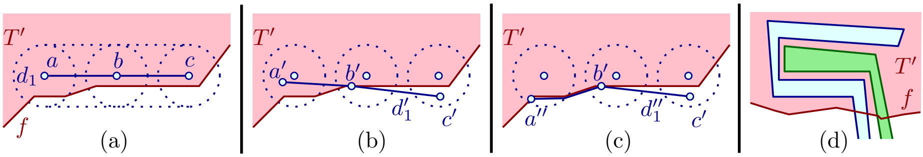


Fig. 5. (a) d_1 is a line segment of a triangle in P_k. f is an image curve of the new triangle. (b) d_1 was originally mapped to d_1' but after adding the image curve f_1 we need to update its mapping. (c) The part of d_1' crosses f into T' can be mapped to f and the remainder left unchanged to form d_1''. (d) The portion of $\sigma[\text{polygon}(P_k)]$ that crosses f into T' is collapsed to lie along f.

This method also leads to the portion of $\sigma[\text{polygon}(P_k)]$ which crosses f into T' being collapsed to f while the remainder remains the same. We want it to be the case that by making arbitrarily small perturbations to all of the moved image points the associated image curves can made non-intersecting. This is sufficient since the Fréchet distance is defined as the infimum across all homeomorphisms and the sub-polygon R can be made into a simple polygon with arbitrarily small perturbations to its points.

Specifically, we map each vertex a to the point in the intersection between its neighborhood and f which has the minimum shortest path distance to a from its previous mapping a'. We can prove that this point is unique. This follows from simple shortest path properties but due to lack of space the proof is deferred to the full version of the paper. Likewise, because of the properties of shortest paths the mappings of these vertices cannot be out of order along f, see Figure 5(d). This method of morphing along shortest paths is similar to the one explored in [7].

For every vertex a in $\sigma[\text{polygon}(P_k)]$ we must compute the intersection of the new curve f with $D_\varepsilon(a)$. In the worst case the curve f has complexity $O(n)$ and $\sigma[\text{polygon}(P_k)]$ contains $O(k)$ vertices, all of which need to be updated. The intersection for a single vertex can be computed in time $O(n)$. As mentioned before, if a was previously mapped to a' we want to remap it to the unique point

$a'' \in (f \cap N_a)$ with the minimum length shortest path to a'. We can compute the length of the shortest path in Q from a' to some point on f in $O(\log(n))$ time [10,11]. We can then compute the optimal point $a'' \in f$ in $O(\log(n))$ steps using binary search for a total runtime of $O(\log^2(n))$ with an additional factor $O(n)$ time to compute associated shortest path in Q between a' and a''. Thus a single point can be updated in time $O(n + \log^2(n) + n)$ or, more simply, time $O(n)$. Therefore all $O(k)$ points in $\sigma[\text{polygon}(P_k)]$ can be updated in $O(kn)$ time.

From Lemma 3 we see that if two image curves cross one of their neighborhoods must be crossed by the other image curve. Thus the above update accounts for all crossings between existing image curves and the added curves. In addition, since the end points of a line segment, $d \in P$, are still mapped in their original neighborhoods its shortest path, d', it must be the case that $\delta_F(d, d') \leq \varepsilon$. Thus, for some mapping τ, it is the case that $\delta_F(\text{polygon}(P_k), \tau[\text{polygon}(P_k)]) \leq \varepsilon$. Following the steps outlined in the above inductive proof yields an algorithm to compute such a τ in time $O(kn)$. This completes the proof of Theorem 4.

Running the algorithm for all m vertices in P yields a total runtime of $O(m^2 n)$. This completes the proof of Theorem 2.

5 Future Work

For simple polygons not in the same plane it seems like the partial matching problem could be shown to be approximately decidable by carefully choosing the distance function used. It would be interesting to see if this method can be extended to more general settings or classes of surfaces such as those in [6]. Likewise it may be worth considering different variations of partial Fréchet distance between surfaces.

References

1. Alt, H., Buchin, M.: Can we compute the similarity between surfaces? Discrete and Computational Geometry 43, 78–99 (2010)
2. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. International Journal of Computational Geometry and Applications 5, 75–91 (1995)
3. Buchin, K., Buchin, M., Schulz, A.: Fréchet Distance of Surfaces: Some Simple Hard Cases. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part II. LNCS, vol. 6347, pp. 63–74. Springer, Heidelberg (2010)
4. Buchin, K., Buchin, M., Wang, Y.: Exact algorithm for partial curve matching via the Fréchet distance. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), pp. 645–654 (2009)
5. Buchin, K., Buchin, M., Wenk, C.: Computing the Fréchet distance between simple polygons in polynomial time. In: 22nd Symposium on Computational Geometry (SoCG), pp. 80–87 (2006)
6. Cook IV, A.F., Driemel, A., Har-Peled, S., Sherette, J., Wenk, C.: Computing the Fréchet Distance between Folded Polygons. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 267–278. Springer, Heidelberg (2011)

7. Efrat, A., Guibas, L.J., Har-Peled, S., Mitchell, J.S.B., Murali, T.M.: New similarity measures between polylines with applications to morphing and polygon sweeping. Discrete and Computational Geometry 28(4), 535–569 (2002)
8. Godau, M.: On the Difficulty of Embedding Planar Graphs with Inaccuracies. In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 254–261. Springer, Heidelberg (1995)
9. Godau, M.: On the complexity of measuring the similarity between geometric objects in higher dimensions. PhD thesis, Freie Universität Berlin, Germany (1998)
10. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. Journal of Computer and System Sciences 39(2), 126–152 (1989)
11. Hershberger, J.: A new data structure for shortest path queries in a simple polygon. Inf. Process. Lett. 38(5), 231–235 (1991)

A Polynomial-Time Approximation Scheme for the Geometric Unique Coverage Problem on Unit Squares*

Takehiro Ito[1], Shin-Ichi Nakano[2], Yoshio Okamoto[3],
Yota Otachi[4], Ryuhei Uehara[4], Takeaki Uno[5], and Yushi Uno[6]

[1] Graduate School of Information Sciences, Tohoku University, Japan
takehiro@ecei.tohoku.ac.jp

[2] Department of Computer Science, Gunma University, Japan
nakano@cs.gunma-u.ac.jp

[3] Department of Communication Engineering and Informatics,
University of Electro-Communications, Japan
okamotoy@uec.ac.jp

[4] School of Information Science, JAIST, Japan
{otachi,uehara}@jaist.ac.jp

[5] Principles of Informatics Research Division,
National Institute of Informatics, Japan
uno@nii.ac.jp

[6] Graduate School of Science, Osaka Prefecture University, Japan
uno@mi.s.osakafu-u.ac.jp

Abstract. We give a polynomial-time approximation scheme for the unique unit-square coverage problem: given a set of points and a set of axis-parallel unit squares, both in the plane, we wish to find a subset of squares that maximizes the number of points contained in exactly one square in the subset. Erlebach and van Leeuwen (2008) introduced this problem as the geometric version of the unique coverage problem, and the best approximation ratio by van Leeuwen (2009) before our work was 2. Our scheme can be generalized to the budgeted unique unit-square coverage problem, in which each point has a profit, each square has a cost, and we wish to maximize the total profit of the uniquely covered points under the condition that the total cost is at most a given bound.

1 Introduction

Let $\mathcal{P}$ be a set of points and $\mathcal{D}$ a set of axis-parallel unit squares,[1] both in the plane $\mathbb{R}^2$. For a subset $\mathcal{C} \subseteq \mathcal{D}$ of unit squares, we say that a point $p \in \mathcal{P}$ is *uniquely covered* by $\mathcal{C}$ if there is exactly one square in $\mathcal{C}$ containing p. In the *unique unit-square coverage problem*, we are given a pair $\langle \mathcal{P}, \mathcal{D} \rangle$ of a set $\mathcal{P}$ of

* This work is partially supported by Grant-in-Aid for Scientific Research, and by the Funding Program for World-Leading Innovative R&D on Science and Technology, Japan.

[1] Throughout this paper, a unit square is of side length 1 and is closed, thus has the boundary.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 24–35, 2012.

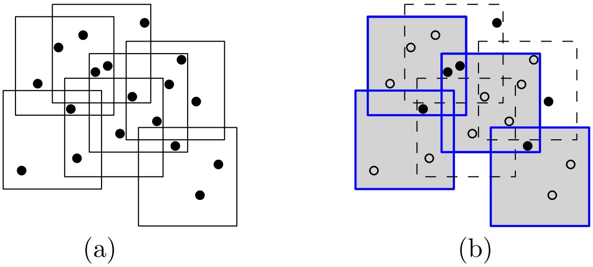


Fig. 1. (a) An instance $\langle \mathcal{P}, \mathcal{D} \rangle$ of the unique unit-square coverage problem and (b) an optimal solution to $\langle \mathcal{P}, \mathcal{D} \rangle$, where each square in the optimal solution is hatched and each uniquely covered point is drawn as a white circle

points and a set $\mathcal{D}$ of axis-parallel unit squares as input, and we are asked to find a subset $\mathcal{C} \subseteq \mathcal{D}$ that maximizes the number of points uniquely covered by $\mathcal{C}$. For example, Fig. 1(b) illustrates an optimal solution to the instance in Fig. 1(a).

In a more general setting, in addition to an instance $\langle \mathcal{P}, \mathcal{D} \rangle$ of the unique unit-square coverage problem, we are given a non-negative real number B, called the *budget*, a non-negative real number $\mathsf{profit}(p)$ for each point $p \in \mathcal{P}$, called the *profit* of p, and a non-negative real number $\mathsf{cost}(S)$ for each square $S \in \mathcal{D}$, called the *cost* of S. In the *budgeted unique unit-square coverage problem*, we are asked to find a subset $\mathcal{C} \subseteq \mathcal{D}$ of total cost at most B such that the total profit of points in $\mathcal{P}$ uniquely covered by $\mathcal{C}$ is maximized. The unique unit-square coverage problem is a specialization of the budgeted unique unit-square coverage problem. To see this, set $\mathsf{profit}(p) = 1$ for all $p \in \mathcal{P}$, $\mathsf{cost}(S) = 0$ for all $S \in \mathcal{D}$, and $B = 0$.

1.1 Past Work and Motivation

Demaine et al. [6] formulated the non-geometric unique coverage problem in more general setting. They gave a polynomial-time $O(\log n)$-approximation algorithm[2] for the non-geometric unique coverage problem, where n is the number of elements (in the geometric version, n corresponds to the number of points). Guruswami and Trevisan [10] studied the same problem and its generalization, which they called the 1-in-k SAT. The unique coverage problem appears in several situations. The previous papers [6,10] provide a connection with unlimited-supply single-minded envy-free pricing. The maximum cut problem can also be modeled as the unique coverage problem. For detail, see their papers.

The parameterized complexity of the unique coverage problem has also been studied. Moser et al. [16] proved that the problem is fixed-parameter tractable when the optimal value is taken as a parameter. Misra et al. [15] later improved the running time.

[2] For notational convenience, throughout the paper, we say that an algorithm for a maximization problem is α-*approximation* if it returns a solution with the objective value APX such that OPT $\leq \alpha$APX, where OPT is the optimal objective value, and hence $\alpha \geq 1$.

Motivated by applications from wireless networks, Erlebach and van Leeuwen [8] studied the geometric versions of the unique coverage problem especially on unit disks. In the context of wireless networks, each point corresponds to a customer location, and the center of each disk corresponds to a place where the provider can build a base station. If several base stations cover a certain customer location, then the resulting interference might cause this customer to receive no service at all. Ideally, each customer should be serviced by exactly one base station. This situation corresponds to the unique unit-disk coverage problem. They showed that the problem on unit disks is strongly NP-hard, and gave a polynomial-time 18-approximation algorithm; for the budgeted unique unit-disk coverage problem, they provided a polynomial-time $(18+\varepsilon)$-approximation algorithm for any fixed constant $\varepsilon > 0$ [8].

The unique unit-square coverage problem is an ℓ_∞ variant (or an ℓ_1 variant) of the unique unit-disk coverage problem. Erlebach and van Leeuwen [8] introduced the budgeted unique unit-square coverage problem, and gave a polynomial-time $(4+\varepsilon)$-approximation algorithm for any fixed constant $\varepsilon > 0$. Later, van Leeuwen [18] gave a proof that the problem on unit squares is also strongly NP-hard, and improved the approximation ratio to $2+\varepsilon$.

Optimization problems on axis-parallel unit squares and unit disks have been thoroughly studied since Huson and Sen [13]. A seminal paper by Hochbaum and Maass [11] established the shifting strategy, which has been used to give a polynomial-time approximation scheme (PTAS) for a lot of problems on unit squares and unit disks (see [12] for example). However, some problems such as coloring [5] and dispersion [9] (see also [7]) are APX-hard already for unit disks. The unique coverage problem is one among the problems for which we know the NP-hardness, but neither APX-hardness nor a PTAS was known. The existence of a PTAS for unit squares has been asked by van Leeuwen [18].

1.2 Contribution of the Paper

In this paper, we give the first PTAS for the unique unit-square coverage problem, and hence we improve the approximation ratio to $1+\varepsilon$ for any fixed constant $\varepsilon > 0$. The algorithm is generalized to give a PTAS for the budgeted unique unit-square coverage problem, too.

We employ the well-known shifting strategy, developed by Baker [1] and applied to the geometric problems by Hochbaum and Maass [11]. Namely, we partition the whole plane into "ribbons" of height one, and delete the points in every $1 + \lceil 1/\varepsilon \rceil$ ribbons. Then, the instance is divided into several subinstances in which all points lie in a rectangle of height $\lceil 1/\varepsilon \rceil$. We compute optimal solutions to such subinstances, and take their union. The best among all choices of possible deletions will be a $(1+\varepsilon)$-approximate solution. On the other hand, van Leeuwen [18] was only able to solve a subinstance in a rectangle of height one, and thus only gave a 2-approximation since he removed the points in every two ribbons.

By the strong NP-hardness, we can conclude that there is no fully polynomial-time approximation scheme unless P = NP [17]; in this sense, a PTAS is the best approximation algorithm for the problem.

Due to the page limitation, all proofs are omitted from this extended abstract.

2 Main Result

The following is the main result of the paper.

Theorem 1. *For any fixed constant $\varepsilon > 0$, there is a polynomial-time $(1+\varepsilon)$-approximation algorithm for the unique unit-square coverage problem.*

We are given an instance $\langle \mathcal{P}, \mathcal{D} \rangle$. Our algorithm consists of two parts. In the first part, we partition the plane into horizontal ribbons of height 1, and show that if there is a polynomial-time exact algorithm for the problem restricted to a constant number of ribbons, then the problem on $\langle \mathcal{P}, \mathcal{D} \rangle$ admits a PTAS. As the second part, Section 3 will be devoted to such a polynomial-time exact algorithm.

A rectangle is *axis-parallel* if its boundary consists of horizontal and vertical line segments. Let R_W be an (unbounded) axis-parallel rectangle of width W and height ∞ which properly contains all points in $\mathcal{P}$ and all unit squares in $\mathcal{D}$. We fix the origin of the coordinate system on the left vertical boundary of R_W. For a square $S \in \mathcal{D}$, we define the (x, y)-coordinates of S as the coordinates of the top right corner of S. We can assume without loss of generality that no horizontal (or vertical) side of a square is on the same line as the horizontal (resp., vertical) side of another square; otherwise, we can scale and translate the squares in polynomial time so that this condition is satisfied [18].

We partition the rectangle R_W into *ribbons* $R_i = [0, W] \times [i, i+1)$, $i \in \mathbb{Z}$, that is, each ribbon is a rectangle of width W and height 1. We may assume without loss of generality that no point in $\mathcal{P}$ and no horizontal side of a square in $\mathcal{D}$ is on the same line as the horizontal boundary of any ribbon [18]. Therefore, every unit square of side length 1 intersects exactly two (consecutive) ribbons. We may assume that each ribbon in R_W contains at least one point in $\mathcal{P}$ and intersects at least one square in $\mathcal{D}$; otherwise, we can simply ignore such ribbons. We thus deal with only a polynomial number of ribbons. For a set G of ribbons, we denote by $\mathcal{P} \cap G$ the set of all points in $\mathcal{P}$ contained in the ribbons in G.

As the first part of our algorithm, we give the following lemma, by applying the well-known shifting strategy [8,11].

Lemma 1. *Let $k = \lceil 1/\varepsilon \rceil$ be a fixed constant, and suppose that we can obtain an optimal solution to $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$ in polynomial time for every set G consisting of at most k ribbons. Then, we can obtain a $(1+\varepsilon)$-approximate solution to $\langle \mathcal{P}, \mathcal{D} \rangle$ in polynomial time.*

3 Algorithm for a Constant Number of Ribbons

The following lemma completes the proof of Theorem 1.

Lemma 2. *The unique unit-square coverage problem on $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$ can be optimally solved in polynomial time for a set G consisting of at most k ribbons, where k is a constant.*

The proof of Lemma 2 is constructive, namely, we give such an algorithm.

3.1 Basic Idea of Our Algorithm

Our algorithm employs a dynamic programming approach based on the line-sweep paradigm. Namely, we look at points and squares from left to right, and extend the uniquely covered region sequentially. However, adding one square S at the rightmost position can influence a lot of squares that were already chosen, and can change the situation drastically (we say that S *influences* a square S' if the region uniquely covered by S' changes after the addition of S). We therefore need to keep track of the squares that are possibly influenced by a newly added square. Unless the number of those squares is bounded by some constant (or the logarithm of the input size), this approach cannot lead to a polynomial-time algorithm. Unfortunately, new squares may influence a super-constant (or super-logarithmic) number of squares.

Instead of adding a square at the rightmost position, we add a square S such that the number of squares that were already chosen and influenced by S can be bounded by a constant. Lemmas 3 and 4 state that we can do this for any set of squares, as long as a trivial condition for the square set to be an optimal solution is satisfied. Furthermore, such a square can be found in polynomial time.

3.2 Basic Definitions

We may assume without loss of generality that the set G consists of consecutive ribbons, forming a *group*; otherwise we can simply solve the problem for each group, because those groups have pairwise distance more than 1. Suppose that G consists of k consecutive ribbons $R_{j+1}, R_{j+2}, \ldots, R_{j+k}$ in R_W, ordered from bottom to top, for some integer j. If a square can cover points in $\mathcal{P} \cap G$, then it is totally included in ribbons $R_j, R_{j+1}, \ldots, R_{j+k+1}$. For notational convenience, in the remainder of this section, we assume $j = 0$ without loss of generality. Note that the two ribbons R_0 and R_{k+1} are not in G.

Since no horizontal side of a square is on the same line as the horizontal boundary of any ribbon, if a square in $\mathcal{D}$ intersects G, then it intersects the lower boundary of exactly one ribbon R_i, $i \in \{1, \ldots, k+1\}$. For each $i \in \{1, \ldots, k+1\}$, we denote by $\mathcal{D}_i \subseteq \mathcal{D}$ the subset of all squares in $\mathcal{D}$ intersecting the lower boundary of R_i. Note that the square sets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{k+1}$ form a partition of the squares intersecting G. No square in $\mathcal{D}_i$ intersects any square in $\mathcal{D}_j$ with

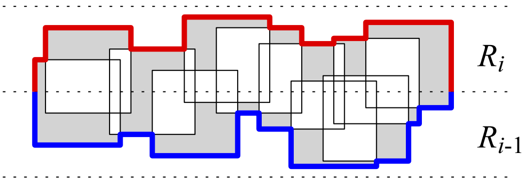


Fig. 2. A set $\mathcal{C}$ of squares in $\mathcal{D}_i$, together with $A_1(\mathcal{C})$ (gray), the upper envelope (red) and the lower envelope (blue). The dotted lines show the lower boundaries of R_{i-1}, R_i and R_{i+1}.

$j \leq i-2$ or $j \geq i+2$. Furthermore, if a square S_i in $\mathcal{D}_i$ intersects a square S_{i+1} in $\mathcal{D}_{i+1}$ (or a square S_{i-1} in $\mathcal{D}_{i-1}$), then the intersection $S_i \cap S_{i+1}$ must be in R_i (resp., $S_{i-1} \cap S_i$ must be in R_{i-1}).

For a square set $\mathcal{C} \subseteq \mathcal{D}$, let $A_0(\mathcal{C})$, $A_1(\mathcal{C})$, $A_2(\mathcal{C})$ and $A_{\geq 3}(\mathcal{C})$ be the areas covered by no square, exactly one square, exactly two squares, and three or more squares in $\mathcal{C}$, respectively. Then, each point contained in the area $A_1(\mathcal{C})$ is uniquely covered by $\mathcal{C}$.

3.3 Properties on Square Subsets of $\mathcal{D}_i$

In this subsection, we deal with squares only in a set $\mathcal{C} \subseteq \mathcal{D}_i$ and the region uniquely covered by them. Of course, squares in $\mathcal{D}_{i-1} \cup \mathcal{D}_{i+1}$ may influence squares in $\mathcal{C}$; this difficulty will be discussed in Section 3.4.

[Upper and lower envelopes]

Let $\mathcal{C} \subseteq \mathcal{D}_i$ be a square set. Since no horizontal (or vertical) side of a square is on the same line as the horizontal (resp., vertical) side of another square, we can partition the boundary of the closure of $A_1(\mathcal{C})$ into two types: the boundary between $A_0(\mathcal{C})$ and $A_1(\mathcal{C})$; and that between $A_1(\mathcal{C})$ and $A_2(\mathcal{C})$. We call the former the *outer boundary* of $\mathcal{C}$. In Fig. 2, the outer boundary of $\mathcal{C}$ is illustrated as (red or blue) thick lines. We call the outer boundary in R_i (or R_{i-1}) the *upper* (resp., *lower*) *envelope* of $\mathcal{C}$. We say that a square S *forms* the boundary of an area A if a part of a side of S is a part of the boundary of the closure of A. Let $UE(\mathcal{C})$ and $LE(\mathcal{C})$ be the sequences of squares that form the upper and lower envelopes of $\mathcal{C}$, from right to left, respectively. Note that a square $S \in \mathcal{C}$ may appear in both $UE(\mathcal{C})$ and $LE(\mathcal{C})$. An example is shown in Fig. 2.

Consider an arbitrary optimal solution $\mathcal{C}^* \subseteq \mathcal{D}_i$ to $\langle \mathcal{P} \cap (R_{i-1} \cup R_i), \mathcal{D}_i \rangle$. If there is a square $S \in \mathcal{C}^*$ that is not part of $A_1(\mathcal{C}^*)$, i.e., $S \cap A_1(\mathcal{C}^*) = \emptyset$, then we can simply remove it from $\mathcal{C}^*$ without losing the optimality. Thus, hereafter we deal with a square set $\mathcal{C} \subseteq \mathcal{D}_i$ such that every square S in $\mathcal{C}$ forms the outer boundary of $\mathcal{C}$, that is, $S \in UE(\mathcal{C})$ or $S \in LE(\mathcal{C})$ holds. This property enables us to extend the upper and lower envelopes sequentially.

[Top squares and the key lemma]

When we add a "new" square S to the current square set $\mathcal{C} \setminus \{S\}$, we need to know the symmetric difference of $A_1(\mathcal{C})$ and $A_1(\mathcal{C} \setminus \{S\})$: the area $A_1(\mathcal{C}) \setminus A_1(\mathcal{C} \setminus \{S\}) \subseteq A_1(\mathcal{C})$ is the uniquely covered area obtained newly by adding S, and the area $A_1(\mathcal{C} \setminus \{S\}) \setminus A_1(\mathcal{C}) \subseteq A_2(\mathcal{C})$ is the non-uniquely covered area due to S. However, it suffices to know $A_1(\mathcal{C} \setminus \{S\}) \setminus A_1(\mathcal{C})$ and its boundary since the boundary of $A_1(\mathcal{C}) \setminus A_1(\mathcal{C} \setminus \{S\})$ is formed only by S and squares forming the boundary of $A_1(\mathcal{C} \setminus \{S\}) \setminus A_1(\mathcal{C})$.

For a square S in a set $\mathcal{C} \subseteq \mathcal{D}$, let $\overline{\Delta}(\mathcal{C}, S)$ be the area $A_1(\mathcal{C} \setminus \{S\}) \setminus A_1(\mathcal{C})$, and $\Delta(\mathcal{C}, S)$ be the set of all squares in $\mathcal{C}$ that form the boundary of $\overline{\Delta}(\mathcal{C}, S)$. An example is shown in Fig. 3. Clearly, every square in $\Delta(\mathcal{C}, S)$ has non-empty intersection with S. We denote by $U\Delta(\mathcal{C}, S)$ the set of all squares that form the boundary of $\overline{\Delta}(\mathcal{C}, S) \cap R_i$, and by $L\Delta(\mathcal{C}, S)$ the set of all squares that form

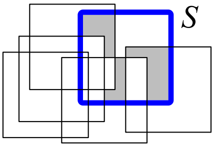


Fig. 3. The gray region shows $\overline{\Delta}(\mathcal{C}, S)$ for the thick square S

the boundary of $\overline{\Delta}(\mathcal{C}, S) \cap R_{i-1}$. As we mentioned in Section 3.1, $\Delta(\mathcal{C}, S)$ may contain a super-constant (or super-logarithmic) number of squares if we simply choose the rightmost square S in $\mathcal{C}$. We will show that, for any square set $\mathcal{C} \subseteq \mathcal{D}_i$, there always exists a square $S \in \mathcal{C}$ such that $\Delta(\mathcal{C}, S)$ contains at most 16 squares, called top squares, and S itself is a top square.

For a square set $\mathcal{C} \subseteq \mathcal{D}_i$, a square $S \in \mathcal{C}$ is called a *top square* of $\mathcal{C}$ if one of the following conditions (i)–(iv) holds:

(i) S is one of the first six squares of $UE(\mathcal{C})$;
(ii) S is one of the first six squares of $LE(\mathcal{C})$;
(iii) S is one of the first two squares of $UE(LE(\mathcal{C}) \setminus UE(\mathcal{C}))$; and
(iv) S is one of the first two squares of $LE(UE(\mathcal{C}) \setminus LE(\mathcal{C}))$.

An example is given in Fig. 4. Remember that the squares in $UE(\mathcal{C})$ and $LE(\mathcal{C})$ are ordered from right to left. We denote by $\mathsf{Top}(\mathcal{C})$ the set of top squares of $\mathcal{C}$. Note that a square may satisfy more than one of the conditions above; indeed, there is no square set $\mathcal{C} \subseteq \mathcal{D}_i$ such that $|\mathsf{Top}(\mathcal{C})| = 16$ since the rightmost square in $\mathcal{C}$ always satisfies both (i) and (ii).

A square set $\mathcal{T} \subseteq \mathcal{D}_i$ is *feasible on* $\mathcal{D}_i$ if $\mathsf{Top}(\mathcal{T}) = \mathcal{T}$. For a feasible square set $\mathcal{T} \subseteq \mathcal{D}_i$, we denote by $\mathfrak{C}_i(\mathcal{T})$ the set of all square subsets of $\mathcal{D}_i$ whose top squares are equal to $\mathcal{T}$, that is,

$$\mathfrak{C}_i(\mathcal{T}) = \{\mathcal{C} \subseteq \mathcal{D}_i \mid \mathsf{Top}(\mathcal{C}) = \mathcal{T}\}.$$

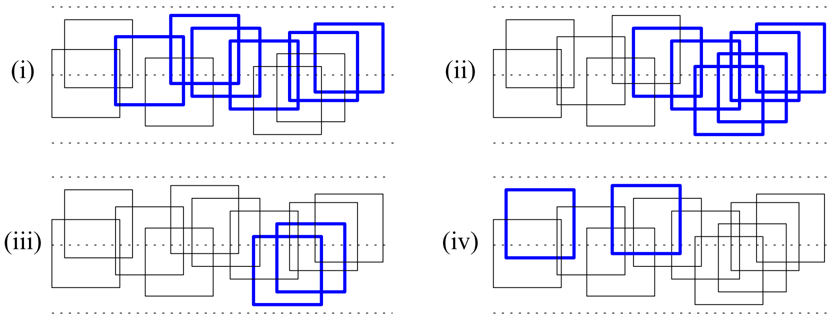


Fig. 4. An example of top squares. The (blue) thick squares are top squares, and the numbers correspond to the conditions in the definition.

A top square S in a feasible set $\mathcal{T}$ is said to be *stable in* $\mathcal{T}$ if $\Delta(\mathcal{C}, S)$ consists only of top squares in $\mathcal{T}$ for any square set $\mathcal{C} \in \mathfrak{C}_i(\mathcal{T})$. Indeed, stable top squares will be crucial to our algorithm: if a top square S is stable in a feasible set $\mathcal{T} \subseteq \mathcal{D}_i$, then $\Delta(\mathcal{C}, S)$ contains at most 16 top squares in $\mathcal{T}$ for any square set $\mathcal{C} \in \mathfrak{C}_i(\mathcal{T})$; and hence we can compute $\Delta(\mathcal{C}, S)$ in polynomial time. Therefore, below is the key lemma for our dynamic programming algorithm.

Lemma 3. *For any feasible square set $\mathcal{T} \subseteq \mathcal{D}_i$, there always exists a top square $K(\mathcal{T})$ which is stable in $\mathcal{T}$. Moreover, $K(\mathcal{T})$ can be found in polynomial time.*

Proof (*Sketch*). We first claim that a square $S \in \mathcal{T}$ is stable in $\mathcal{T}$ if and only if $S' \notin \Delta(\mathcal{T} \cup \{S'\}, S)$ holds for every square $S' \in \mathcal{D}_i \setminus \mathcal{T}$ such that $\mathsf{Top}(\mathcal{T} \cup \{S'\}) = \mathcal{T}$. This claim implies that we can check in polynomial time whether a square $S \in \mathcal{T}$ is stable in $\mathcal{T}$.

Let S_1 be the square in $\mathcal{T}$ with the largest x-coordinate. If S_1 is stable in $\mathcal{T}$, we set $K(\mathcal{T}) = S_1$. Otherwise, the first or the second square in $LE(\mathcal{T}) \setminus UE(\mathcal{T})$ or in $UE(\mathcal{T}) \setminus LE(\mathcal{T})$ is stable in $\mathcal{T}$.

More specifically, consider the case where S_1 is not stable in $\mathcal{T}$. Then, there exists a non-top square $S' \in \mathcal{D}_i \setminus \mathcal{T}$ such that $S' \in \Delta(\mathcal{T} \cup \{S'\}, S)$ and $\mathsf{Top}(\mathcal{T} \cup \{S'\}) = \mathcal{T}$. If $S' \in U\Delta(\mathcal{T} \cup \{S'\}, S)$, then let S_2 and S_3 be the first and the second squares in $LE(\mathcal{T}) \setminus UE(\mathcal{T})$, respectively. (The case for $S' \in L\Delta(\mathcal{T} \cup \{S'\}, S)$ is symmetric.) If S_3 is in $UE(LE(\mathcal{T}) \setminus UE(\mathcal{T}))$, then S_2 is stable in $\mathcal{T}$ and hence we set $K(\mathcal{T}) = S_2$. Otherwise, S_3 is stable in $\mathcal{T}$ and hence we set $K(\mathcal{T}) = S_3$. □

3.4 Properties on Square Subsets of $\mathcal{D}$

Remember that the ribbons $R_0, R_1, \ldots, R_{k+1}$ are ordered from bottom to top, and that $\mathcal{D}_i$ is the set of all squares in $\mathcal{D}$ intersecting the lower boundary of R_i for each $i \in \{1, \ldots, k+1\}$. For a square set $\mathcal{C} \subseteq \mathcal{D}$, let $\mathcal{C}_i = \mathcal{C} \cap \mathcal{D}_i$ for each $i \in \{1, \ldots, k+1\}$. Then, these square sets $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{k+1}$ form a partition of $\mathcal{C}$.

A square set $\mathcal{T} \subseteq \mathcal{D}$ is *feasible on* $\mathcal{D}$ if $\mathsf{Top}(\mathcal{T} \cap \mathcal{D}_i) = \mathcal{T} \cap \mathcal{D}_i$ for each $i \in \{1, \ldots, k+1\}$. For a feasible square set $\mathcal{T}$ on $\mathcal{D}$ and $i \in \{1, \ldots, k+1\}$, we denote by $\mathcal{T}_i = \mathcal{T} \cap \mathcal{D}_i$, and let

$$\mathfrak{C}(\mathcal{T}) = \{\mathcal{C} \subseteq \mathcal{D} \mid \mathsf{Top}(\mathcal{C}_i) = \mathcal{T}_i \text{ for each } i \in \{1, \ldots, k+1\}\}.$$

We say that $\mathcal{T}_i$ is *safe for* $\mathcal{T}$ if $\Delta(\mathcal{C}, K(\mathcal{T}_i)) \subset \mathcal{T}$ for any square set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$, where $K(\mathcal{T}_i)$ is the stable top square in $\mathcal{T}_i$ which is selected as in the proof of Lemma 3.

Lemma 4. *For any feasible square set $\mathcal{T}$ on $\mathcal{D}$, there exists an index $q \in \{1, \ldots, k+1\}$ such that $\mathcal{T}_q$ is safe for $\mathcal{T}$.*

Proof (*Sketch*). Let $\mathcal{T}$ be a feasible square set on $\mathcal{D}$. Then, for each $i \in \{2, \ldots, k\}$, $\mathcal{T}_{i-1}$, $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$ are feasible square sets on $\mathcal{D}_{i-1}$, $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$, respectively. We say that $\mathcal{T}_i$ is *safe for* $\mathcal{T}_{i+1}$ if $\Delta(\mathcal{C}_i \cup \mathcal{C}_{i+1}, K(\mathcal{T}_i)) \subset \mathcal{T}_i \cup \mathcal{T}_{i+1}$ for any square set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$. Similarly, we say that $\mathcal{T}_i$ is *safe for* $\mathcal{T}_{i-1}$ if $\Delta(\mathcal{C}_{i-1} \cup \mathcal{C}_i, K(\mathcal{T}_i)) \subset \mathcal{T}_{i-1} \cup \mathcal{T}_i$ for any square set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$. For notational convenience, let $\mathcal{D}_0 = \emptyset$ and

$\mathcal{D}_{k+2} = \emptyset$; $\mathcal{T}_1$ is always safe for $\mathcal{T}_0$; and $\mathcal{T}_{k+1}$ is always safe for $\mathcal{T}_{k+2}$. Since each ribbon is of height 1 and each square is of side length 1, the square $K(\mathcal{T}_i) \in \mathcal{D}_i$ intersects only squares in $\mathcal{D}_{i-1}$ and $\mathcal{D}_{i+1}$. Therefore, for $i \in \{1, \ldots, k+1\}$, $\mathcal{T}_i$ is safe for $\mathcal{T}$ if and only if $\mathcal{T}_i$ is safe for both $\mathcal{T}_{i-1}$ and $\mathcal{T}_{i+1}$. Thus, it suffices to show that one of $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$ is safe for the other for each $i \in \{1, \ldots, k\}$. Then, there exists an index $q \in \{1, \ldots, k+1\}$ such that $\mathcal{T}_q$ is safe for both $\mathcal{T}_{q-1}$ and $\mathcal{T}_{q+1}$.

Let $\mathcal{C}$ be a square set in $\mathfrak{C}(\mathcal{T})$. For each $i \in \{1, \ldots, k+1\}$, let $ux(\mathcal{C}_i)$ be the x-coordinate of the leftmost point of the area $R_i \cap K(\mathcal{T}_i) \cap \big(A_1(\mathcal{C}_i) \cup A_2(\mathcal{C}_i)\big)$, while let $lx(\mathcal{C}_i)$ be the x-coordinate of the leftmost point of the area $R_{i-1} \cap K(\mathcal{T}_i) \cap \big(A_1(\mathcal{C}_i) \cup A_2(\mathcal{C}_i)\big)$. Since $A_1(\mathcal{C}_i) \cap S \neq \emptyset$ for every square $S \in \mathcal{C}_i$, both $ux(\mathcal{C}_i)$ and $lx(\mathcal{C}_i)$ are well-defined. Since $K(\mathcal{T}_i)$ is stable in $\mathcal{T}_i$, we see that $ux(\mathcal{C}_i)$ is invariant under the choice of $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$. Thus, we also write $ux(\mathcal{T}_i)$ to mean $ux(\mathcal{C}_i)$ for any set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$. The same applies to $lx(\mathcal{T}_i)$. We claim that

(a) $\mathcal{T}_i$ is safe for $\mathcal{T}_{i+1}$ if $lx(\mathcal{T}_{i+1}) < ux(\mathcal{T}_i)$; and

(b) $\mathcal{T}_{i+1}$ is safe for $\mathcal{T}_i$ if $ux(\mathcal{T}_i) < lx(\mathcal{T}_{i+1})$.

Since no vertical side of a square is on the same line as the vertical side of another square, $ux(\mathcal{T}_i) \neq lx(\mathcal{T}_{i+1})$ for each $i \in \{1, \ldots, k\}$. Therefore, at least one of $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$ is safe for the other. □

3.5 Algorithm for the Problem on $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$

We are now ready to describe our algorithm for the problem on $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$.

For a feasible square set $\mathcal{T}$ on $\mathcal{D}$, let $f(\mathcal{T})$ be the maximum number of points in $\mathcal{P} \cap G$ uniquely covered by a square set in $\mathfrak{C}(\mathcal{T})$, that is,

$$f(\mathcal{T}) = \max\{\mathsf{profit}(\mathcal{P} \cap G, \mathcal{C}) \mid \mathcal{C} \in \mathfrak{C}(\mathcal{T})\},$$

where $\mathsf{profit}(\mathcal{P} \cap G, \mathcal{C})$ is the number of points in $\mathcal{P} \cap G$ that are uniquely covered by $\mathcal{C}$. Then, the optimal value $\mathrm{OPT}(\mathcal{P} \cap G, \mathcal{D})$ for $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$ can be computed as

$$\mathrm{OPT}(\mathcal{P} \cap G, \mathcal{D}) = \max\{f(\mathcal{T}) \mid \mathcal{T} \text{ is feasible on } \mathcal{D}\}.$$

Since $|\mathcal{T}| < 16(k+1)$, this computation can be done in polynomial time if we have the values $f(\mathcal{T})$ for all feasible square sets $\mathcal{T}$ on $\mathcal{D}$.

We thus compute $f(\mathcal{T})$ in polynomial time for all feasible square sets $\mathcal{T}$ on $\mathcal{D}$, according to the "parent-child relation." For a square set $\mathcal{C} \subseteq \mathcal{D}$, we denote simply by $\mathsf{Top}(\mathcal{C}) = \bigcup_{1 \le i \le k+1} \mathsf{Top}(\mathcal{C}_i)$. For a feasible square set $\mathcal{T}$ on $\mathcal{D}$, let $K(\mathcal{T}) = K(\mathcal{T}_q)$ where $\mathcal{T}_q = \mathcal{T} \cap \mathcal{D}_q$ is safe for $\mathcal{T}$. For two feasible square sets $\mathcal{T}$ and $\mathcal{T}'$ on $\mathcal{D}$, we say that $\mathcal{T}'$ is a *child* of $\mathcal{T}$ if there exists a square set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$ such that $\mathsf{Top}(\mathcal{C} \setminus \{K(\mathcal{T})\}) = \mathcal{T}'$.

Lemma 5. *The parent-child relation for the feasible square sets on $\mathcal{D}$ can be constructed in polynomial time. Furthermore, the parent-child relation is acyclic.*

We finally give the algorithm that solves the problem on $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$.

For each $i \in \{1, \ldots, k+1\}$, let $\mathcal{D}_i^0$ be the square set consisting of the first 16 squares in $\mathcal{D}_i$ having the smallest x-coordinates. Let $\mathcal{D}^0 = \bigcup_{1 \le i \le k+1} \mathcal{D}_i^0$, then

$|\mathcal{D}^0| \le 16(k+1)$. As the initialization, we first compute $f(\mathcal{T})$ for all feasible sets $\mathcal{T}$ on $\mathcal{D}^0$. Since $|\mathcal{D}^0|$ is a constant, the total number of feasible sets $\mathcal{T}$ on $\mathcal{D}^0$ is also a constant. Therefore, this initialization can be done in polynomial time.

We then compute $f(\mathcal{T})$ for a feasible square set $\mathcal{T}$ on $\mathcal{D}$ from the values $f(\mathcal{T}')$ for all children $\mathcal{T}'$ of $\mathcal{T}$. Since the parent-child relation is acyclic, we can find a feasible square set $\mathcal{T}$ such that the values $f(\mathcal{T}')$ are already computed for all children $\mathcal{T}'$ of $\mathcal{T}$. By Lemma 4 there always exists a feasible square set $\mathcal{T}_q = \mathcal{T} \cap \mathcal{D}_q$ on $\mathcal{D}_q$ which is safe for $\mathcal{T}$, and hence by Lemma 3 we have a stable top square $K(\mathcal{T}) = K(\mathcal{T}_q)$ in polynomial time. For a square set $\mathcal{C} \subseteq \mathcal{D}$ and a square $S \in \mathcal{C}$, we denote by $z(\mathcal{C}, S)$ the difference of the number of uniquely covered points in $\mathcal{P} \cap G$ caused by adding S to $\mathcal{C} \setminus \{S\}$, that is, the number of points in $\mathcal{P} \cap G$ that are included in $S \cap A_1(\mathcal{C})$ minus the number of points in $\mathcal{P} \cap G$ that are included in $S \cap A_1(\mathcal{C} \setminus \{S\})$. Since $\mathcal{T}_q$ is safe for $\mathcal{T}$ and $K(\mathcal{T}) = K(\mathcal{T}_q)$, we have $z(\mathcal{T}, K(\mathcal{T})) = z(\mathcal{C}, K(\mathcal{T}))$ for all square sets $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$. Therefore, we can correctly update $f(\mathcal{T})$ by

$$f(\mathcal{T}) := \max\{f(\mathcal{T}') \mid \mathcal{T}' \text{ is a child of } \mathcal{T}\} + z(\mathcal{T}, K(\mathcal{T})). \tag{1}$$

This way, the algorithm correctly solves the problem on $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$ in polynomial time.

This completes the proof of Lemma 2. □

4 Budgeted Version

In this section, we give the following theorem.

Theorem 2. *For any fixed constant* $\varepsilon > 0$*, there is a polynomial-time* $(1+\varepsilon)$*-approximation algorithm for the budgeted unique unit-square coverage problem.*

We give a sketch how to adapt the algorithm above to the budgeted unique unit-square coverage problem. To this end, we first describe the adaptation to give an optimal solution to $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$ in pseudo-polynomial time when budget, cost, and profit are all integers.

We keep the same strategy, but for the dynamic programming, we slightly change the definition of f. In the budgeted version, $\mathsf{profit}(\mathcal{P}, \mathcal{C})$ means the total profit of the points in $\mathcal{P}$ that are uniquely covered by $\mathcal{C}$, and $\mathsf{cost}(\mathcal{C})$ means the total cost of the squares in $\mathcal{C}$. Let $X = \sum_{p \in \mathcal{P}} \mathsf{profit}(p)$, then $\mathsf{profit}(\mathcal{P}, \mathcal{C}) \le X$ for any square set $\mathcal{C} \subseteq \mathcal{D}$. For a feasible square set $\mathcal{T} \subseteq \mathcal{D}$ and an integer $x \in \{0, \dots, X\}$, let $g(\mathcal{T}, x)$ be the minimum total cost of squares in a set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$ such that the total profit of uniquely covered points in $\mathcal{P} \cap G$ by $\mathcal{C}$ is at least x, that is,

$$g(\mathcal{T}, x) = \min\{\mathsf{cost}(\mathcal{C}) \mid \mathcal{C} \in \mathfrak{C}(\mathcal{T}) \text{ and } \mathsf{profit}(\mathcal{P} \cap G, \mathcal{C}) \ge x\}.$$

If there is no square set $\mathcal{C} \in \mathfrak{C}(\mathcal{T})$ such that $\mathsf{profit}(\mathcal{P} \cap G, \mathcal{C}) \ge x$, then let $g(\mathcal{T}, x) = +\infty$. Then, the optimal value $\mathrm{OPT}(\mathcal{P} \cap G, \mathcal{D})$ for the budgeted version on $\langle \mathcal{P} \cap G, \mathcal{D} \rangle$ can be computed as

$$\mathrm{OPT}(\mathcal{P} \cap G, \mathcal{D}) = \max\{x \mid 0 \le x \le X, g(\mathcal{T}, x) \le B\}.$$

We proceed along the same way as the algorithm in Section 3.5, except for the update formula (1) that should be replaced by

$$g(\mathcal{T},x):=\min\{g(\mathcal{T}',y) \mid \mathcal{T}' \text{ is a child of } \mathcal{T},\ y+z(\mathcal{T},K(\mathcal{T}))\geq x\}+\mathsf{cost}(K(\mathcal{T})),$$

where $z(\mathcal{T},K(\mathcal{T}))$ means the difference of the total profit of uniquely covered points in $\mathcal{P}\cap G$ caused by adding the square $K(\mathcal{T})$ to $\mathcal{T}\setminus\{K(\mathcal{T})\}$. This way, we obtain an optimal solution to $\langle\mathcal{P}\cap G,\mathcal{D}\rangle$ for a group G consisting of at most k consecutive ribbons. Note that the blowup in the running time is only polynomial in X.

Let $R_1,R_2,\dots,R_t$ be the ribbons in R_W ordered from bottom to top. For each $j\in\{0,\dots,k\}$, let R_W^j be the set of groups $G_1,G_2,\dots$, each of which consists of at most k ribbons, obtained from R_W by deleting the ribbons R_i if and only if $i=j \bmod k+1$. We now explain how to obtain a solution to the problem on $\langle\mathcal{P}\cap R_W^j,\mathcal{D}\rangle$. The adapted algorithm above can solve the problem on each group G_l in R_W^j, and hence suppose that we have computed $g(\mathcal{T},x)$ for each group G_l and all integers $x\in\{0,\dots,X\}$. Then, obtaining a solution to $\langle\mathcal{P}\cap R_W^j,\mathcal{D}\rangle$ can be regarded as solving an instance of the multiple-choice knapsack problem [4,14]. The multiple-choice knapsack problem can be solved in pseudo-polynomial time which polynomially depends on X [4,14], and hence we can obtain an optimal solution to $\langle\mathcal{P}\cap R_W^j,\mathcal{D}\rangle$, $0\leq j\leq k$, in pseudo-polynomial time.

Then, by the standard scale-and-round technique (as used for the ordinary knapsack problem) [4,14], for any fixed constant $\varepsilon'>0$, we obtain a $(1+\varepsilon')$-approximate solution to $\langle\mathcal{P}\cap R_W^j,\mathcal{D}\rangle$ for each $j\in\{0,\dots,k\}$. Overall, we can obtain such an approximate solution to each of the $k+1$ subinstances $\langle\mathcal{P}\cap R_W^j,\mathcal{D}\rangle$, $0\leq j\leq k$, in polynomial time, and taking the best one gives a PTAS for the budgeted unique unit-square coverage problem on $\langle\mathcal{P},\mathcal{D}\rangle$.

5 Conclusion

The PTAS in this paper combines the well-known shifting strategy [1,11] and a novel dynamic programming algorithm to solve the problem restricted to regions of constant height, and answers a question by van Leeuwen [18]. The generality of the approach enables us to solve the budgeted version, too.

The running time of our PTAS is a polynomial of degree depending on ε. It is desirable to obtain a PTAS such that the degree of its polynomial running time does not depend on ε: such a PTAS is called an efficient PTAS (EPTAS). The existence of an EPTAS would be excluded by showing W[1]-hardness (unless FPT = W[1]) [2,3], but the unique coverage problem is fixed-parameter tractable [15,16], thus unlikely to be W[1]-hard. The existence of an EPTAS is left open.

References

1. Baker, B.: Approximation algorithms for NP-complete problems on planar graphs. J. ACM 41, 153–180 (1994)
2. Bazgan, C.: Schémas d'approximation et complexité paramétrée. Rapport de DEA, Université Paris Sud (1995)

3. Cesati, M., Trevisan, L.: On the efficiency of polynomial time approximation schemes. Information Processing Letters 64, 165–171 (1997)
4. Chandra, A.K., Hirschberg, D.S., Wong, C.K.: Approximate algorithms for some generalized knapsack problems. Theoretical Computer Science 3, 293–304 (1976)
5. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Mathematics 86, 165–177 (1990)
6. Demaine, E.D., Hajiaghayi, M.T., Feige, U., Salavatipour, M.R.: Combination can be hard: approximability of the unique coverage problem. SIAM J. on Computing 38, 1464–1483 (2008)
7. Dumitrescu, A., Jiang, M.: Dispersion in unit disks. In: Proc. STACS 2010, pp. 299–310 (2010)
8. Erlebach, T., van Leeuwen, E.J.: Approximating geometric coverage problems. In: Proc. SODA 2008, pp. 1267–1276 (2008)
9. Fiala, J., Kratochvíl, J., Proskurowski, A.: Systems of distant representatives. Discrete Applied Mathematics 145, 306–316 (2005)
10. Guruswami, V., Trevisan, L.: The Complexity of Making Unique Choices: Approximating 1-in-k SAT. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 99–110. Springer, Heidelberg (2005)
11. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32, 130–136 (1985)
12. Hunt III, H.B., Marathe, M.V., Radhakrishnan, V., Ravi, S.S., Rosenkrantz, D.J., Stearns, R.E.: NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. J. Algorithms 26, 238–274 (1998)
13. Huson, M.L., Sen, A.: Broadcast scheduling algorithms for radio networks. In: Proc. IEEE MILCOM 1995, pp. 647–651 (1995)
14. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
15. Misra, N., Raman, V., Saurabh, S., Sikdar, S.: The Budgeted Unique Coverage Problem and Color-Coding. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 310–321. Springer, Heidelberg (2009)
16. Moser, H., Raman, V., Sikdar, S.: The Parameterized Complexity of the Unique Coverage Problem. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 621–631. Springer, Heidelberg (2007)
17. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
18. van Leeuwen, E.J.: Optimization and approximation on systems of geometric objects. Ph.D. Thesis, University of Amsterdam (2009)

Watchman Routes for Lines and Segments

Adrian Dumitrescu[1], Joseph S.B. Mitchell[2], and Paweł Żyliński[3]

[1] Dept. of Computer Science, Univ. of Wisconsin–Milwaukee, USA
[2] Dept. of Appl. Math. and Statistics, State Univ. of NY at Stony Brook, USA
[3] Inst. of Informatics, University of Gdańsk, Poland

Abstract. Given a set $\mathcal{L}$ of non-parallel lines, a watchman route (tour) for $\mathcal{L}$ is a closed curve contained in the union of the lines in $\mathcal{L}$ such that every point on any line is visible (along a line) from at least one point of the route; similarly, we define a watchman route (tour) for a connected set $\mathcal{S}$ of line segments. The watchman route problem for a given set of lines or line segments is to find a shortest watchman route for the input set, and these problems are natural special cases of the watchman route problem in multiply connected polygonal domains.

In this paper, we show that the problem of computing a shortest watchman route for a set of n non-parallel lines in the plane is polynomially tractable, while it becomes NP-hard in 3D. Then, we reprove NP-hardness of this problem for line segments in the plane and provide a polynomial-time approximation algorithm with ratio $O(\log^3 n)$. Additionally, we consider some special cases of the watchman route problem on line segments, for which we provide improved approximation or exact algorithms.

1 Introduction

In 1973, Victor Klee asked for the minimum number of stationary guards that can watch over all the paintings that hang in a gallery with n walls. The first answer was given by Chvátal [4], who proved that $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and sometimes necessary to cover a polygon with n vertices. Over the last few decades, numerous variations of the above *art gallery problem* have been studied, including mobile guards, guards with limited visibility or mobility, guarding special classes of polygons, etc.; see O'Rourke's monograph [17] and the survey articles [19,25].

The watchman route problem for polygons has been introduced in [2,3]. A *watchman route* in a polygon is a closed curve inside the polygon such that every point in the polygon is visible from at least one point of the route. For simple polygons, the shortest route can be found in $O(n^4 \log n)$ time [5,21], and a 2-approximation can be computed in linear time [22]. For polygons with holes the problem is NP-hard [3,7]. An $O(\log n)$-approximation algorithm for the special case of orthogonal polygons (and orthogonal visibility) has been reported in [14]. No approximation algorithms are known for the general watchman route problem in polygons with holes.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 36–47, 2012.

In this paper, we study a natural special case of the watchman route problem in polygons with holes. We consider watchman routes for a collection of lines or for a collection of line segments. One can view the lines or the line segments to be streets in a city. A watchman route is constrained to lie on the road network, i.e., the union of the lines or line segments. A line (or a line segment) can be "seen" in both directions from any point incident to it, in particular, from any such vertex of the arrangement of lines (or line segments). Consequently, a watchman route for a collection of lines (or line segments) is a polygonal route. See Fig. 1 for an illustration.

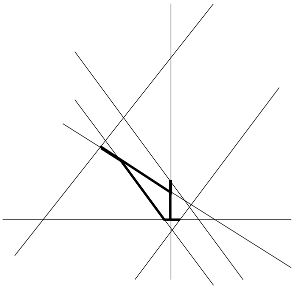


Fig. 1. A watchman route (in bold) for a set of seven lines (or line segments)

A set of lines $\mathcal{L}$ is said to be *connected* if there exists a path $\xi \subset \cup_{l \in \mathcal{L}} l$ from any point $p \in l$ to any other point $p' \in l'$, for any $l, l' \in \mathcal{L}$. Similarly, we define a connected set of segments. Observe that a set of lines $\mathcal{L}$ is connected if and only if not all lines are parallel (i.e., there exist two non-parallel lines in $\mathcal{L}$). Formally, the watchman route problem for lines or line segments in the plane is defined as follows.

The watchman route problem for lines (WRL): Given a set $\mathcal{L}$ of non-parallel lines in the plane, find a shortest watchman route for $\mathcal{L}$.

The watchman route problem for segments (WRS): Given a connected set $\mathcal{S}$ of line segments in the plane, find a shortest watchman route for $\mathcal{S}$.

To the best of our knowledge, only the WRS problem has been previously considered for arrangements of axis-aligned segments, so-called *grids* [26]; this variant has been introduced by Frank Hoffmann and so has become known as "Frank's problem" [12]. Xu and Brass [26] prove the NP-hardness of the WRS problem by a reduction from the connected vertex cover problem in planar graphs with maximum degree four. Other variants of the art gallery problem for line segments have been studied in [1,11,13,16,17,23,24], to mention just a few.

Our Results. In Section 2, we provide a polynomial-time algorithm for computing a shortest watchman route for a set of n non-parallel lines in the plane, and show that the watchman route problem for orthogonal lines in 3D is NP-hard.

Next, in Section 3, we show that the watchman route problem for axis-aligned line segments in the plane is also NP-hard (with a simpler proof than [26]), and give an approximation algorithm with ratio $O(\log^3 n)$ for the case of any connected set of segments in any dimension. Additionally, we then show that an approximation algorithm with a constant ratio exists for certain special cases of the watchman route problem for segments, and discuss how to compute an optimal watchman route for "simple arrangements" of segments and for "almost simple grids."

A connected set of lines or segments can be thought of as a polygon with holes, consisting of very thin corridors. Thus in particular, our result for lines provides a polynomial time optimal algorithm for the watchman route problem in a restricted subclass of polygons with holes, while the result for line segments provides a polynomial time $O(\log^3 n)$-approximation algorithm for another wider subclass of polygons with holes; recall that the approximation algorithm in [14] applies only to orthogonal polygons with holes, under orthogonal visibility. In addition, our result for lines shows that some instances of TSP with neighborhoods (TSPN) *and with obstacles* are polynomially solvable (the obstacles are the open faces of the input line arrangement), while obviously TSPN with obstacles is generally NP-hard. It is worth mentioning that TSPN for a set of lines in the plane (with no obstacles) is solvable in polynomial time [6].

Notation. Throughout this paper we use the following notations. If G is a graph drawn in the plane, and $s, t \in V(G)$, let $\pi(s,t) = \pi_G(s,t)$ denote a shortest path connecting s and t in G, and let $|\pi(s,t)|$ denote its length. Next, for a set of lines $\mathcal{L}$ (resp. a set of line segments $\mathcal{S}$), let $V(\mathcal{A}(\mathcal{L}))$ (resp. $V(\mathcal{A}(\mathcal{S}))$) denote the set of vertices of the arrangement $\mathcal{A}(\mathcal{L})$ (resp. $\mathcal{A}(\mathcal{S})$) formed by the lines in $\mathcal{L}$ (resp. line segments in $\mathcal{S}$). Let $G(\mathcal{L})$ (resp. $G(\mathcal{S})$) denote the weighted planar graph with vertex set $V(\mathcal{A}(\mathcal{L}))$ (resp. $V(\mathcal{A}(\mathcal{S}))$) whose edges connect successive vertices on the lines in $\mathcal{L}$ (resp. the line segments in $\mathcal{S}$); the weight of an edge is the Euclidean distance between the corresponding vertices along the connecting line (resp. segment). Finally, for a set of lines $\mathcal{L}$ (resp. a set of line segments $\mathcal{S}$), let $OPT(\mathcal{L})$ (resp. $OPT(\mathcal{S})$) denote an optimal watchman route for $\mathcal{L}$ (resp. $\mathcal{S}$), and $\mathrm{OPT} = |OPT(\mathcal{L})|$ (resp. $\mathrm{OPT} = |OPT(\mathcal{S})|$) be its length.

2 The Watchman Route Problem for Lines

In this section, we first provide a polynomial-time algorithm for computing a shortest watchman route for a set $\mathcal{L}$ of n lines in the plane. Our approach is based upon a dynamic programming technique. At the end of this section we shortly discuss the 3D case, namely, we show that the watchman route problem for lines in 3D becomes NP-hard, even for axis-parallel lines.

2.1 Lines in the Plane

The input is a set $\mathcal{L}$ of n lines, not all parallel to each other; otherwise, the line arrangement is not connected, and no watchman route exists. We can assume w.l.o.g. that no line in $\mathcal{L}$ is horizontal. Our algorithm is based on the following crucial observation: Since a watchman route is connected, a line $\ell \in \mathcal{L}$ intersects a watchman route R if and only if it intersects the convex hull of R. Thus, in order to compute an optimal watchman tour $OPT(\mathcal{L})$ for $\mathcal{L}$, we only need to compute the convex hull of $OPT(\mathcal{L})$. This is done by solving the *minimum convex hull* problem, defined as follows (see Fig. 2).

The minimum convex hull problem (MCH): Given a set $\mathcal{L}$ of n lines in the plane, not all parallel, compute a minimum-length cyclic sequence $(v_1, \ldots, v_h, v_1)$ of vertices $v_i \in V(\mathcal{A}(\mathcal{L}))$ in convex position, such that every line in $\mathcal{L}$ intersects the convex polygon $v_1, \ldots, v_h$, where the *length of* $(v_1, \ldots, v_h, v_1)$ is defined to be $\sum_{i=1}^{h} |\pi(v_i, v_{i+1})|$, with $v_{h+1} = v_1$.

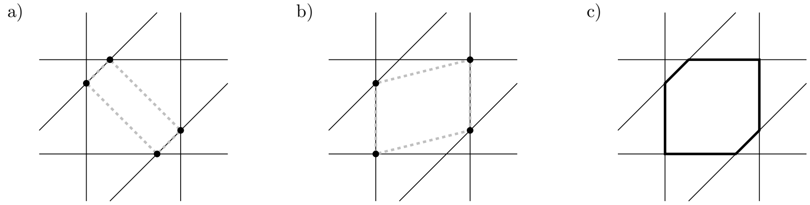


Fig. 2. The MCH problem. Observe that there may be two or more optimal solutions of MCH (left, middle) yielding the same optimal watchman tour (right).

The following lemma formalizes the relationship between the MCH problem and the watchman route problem for lines (the WRL problem).

Lemma 1. *For a set $\mathcal{L}$ of (non-parallel) lines in the plane, a solution to the MCH problem for $\mathcal{L}$ yields a solution to the WRL problem for $\mathcal{L}$.*

Proof. Let $(v_1, \ldots, v_h, v_1)$ be an optimal solution to the MCH problem for $\mathcal{L}$. Consider the route R that results from the concatenation of the shortest paths $\pi(v_1, v_2), \ldots, \pi(v_{h-1}, v_h)$, and $\pi(v_h, v_1)$. We claim that R is a shortest watchman route for $\mathcal{L}$.

First, since each line $\ell \in \mathcal{L}$ intersects the convex polygon $v_1, \ldots, v_h$, each line $\ell \in \mathcal{L}$ must intersect the route R. (Otherwise, a line ℓ would separate some vertex v_i from some other vertex v_j, without intersecting R — a contradiction to the connectedness of R.) Thus, R is a watchman route for $\mathcal{L}$.

Note that a cyclic sequence $C = (v_1, \ldots, v_h, v_1)$ optimizing MCH, i.e., the convex polygon $Q = v_1, \ldots, v_h$, may be strictly contained in the convex hull Q' of the corresponding route (obtained by using shortest paths to link the vertices of C); however, in this case, there are multiple optimal solutions to the MCH problem — Q' also induces a solution to the MCH problem, with the same length as C.

Next, consider an optimal route $OPT(\mathcal{L})$, which is a solution to the WRL problem for $\mathcal{L}$. Since the vertices of $OPT(\mathcal{L})$ are vertices of the arrangement

$\mathcal{A}(\mathcal{L})$, we know that the convex hull $\mathrm{CH}(OPT(\mathcal{L}))$ of the route has vertices in the set $V(\mathcal{A}(\mathcal{L}))$. Since each $\ell \in \mathcal{L}$ intersects the route $OPT(\mathcal{L})$, we know that each $\ell \in \mathcal{L}$ also intersects $\mathrm{CH}(OPT(\mathcal{L}))$. Thus, the vertices of $\mathrm{CH}(OPT(\mathcal{L}))$ form a cyclic sequence of vertices in $V(\mathcal{A}(\mathcal{L}))$ that is feasible for the MCH problem, and the length of this cyclic sequence is exactly the length of $OPT(\mathcal{L})$, since $OPT(\mathcal{L})$ must use a shortest path to link any two consecutive vertices of $\mathrm{CH}(OPT(\mathcal{L}))$ (otherwise, the route could be shortened while still visiting every line of $\mathcal{L}$). □

Observe that there may be several optimal solutions to the MCH problem, all having the same length but having different sequences of vertices in convex position, that all correspond to the same optimal tour $OPT(\mathcal{L})$.

Dynamic Programming for the MCH Problem. The goal is to find a minimum-length cyclic sequence $C = (v_1, \ldots, v_h, v_1)$, in convex position, whose vertices are in $V(\mathcal{A}(\mathcal{L}))$, such that every line $\ell \in \mathcal{L}$ intersects the convex polygon $Q = v_1, \ldots, v_h$; recall that the *length of* $(v_1, \ldots, v_h, v_1)$ is defined as $\sum_{i=1}^{h} |\pi(v_i, v_{i+1})|$, with $v_{h+1} = v_1$.

First, we handle the trivial cases $h = 1, 2$. It may be that Q is a single point ($h = 1$), a vertex of $\mathcal{A}(\mathcal{L})$; this is trivial to check, since this happens only if all lines of $\mathcal{L}$ pass through one point. It may be that Q corresponds to a line segment ($h = 2$). This case is also trivial to check, since we can enumerate all pairs $\{v_1, v_2\}$ of vertices in $V(\mathcal{A}(\mathcal{L}))$, check for each choice whether it is the case that every $\ell \in \mathcal{L}$ intersects the (closed) line segment $v_1 v_2$, and report a pair having the shortest length $|\pi(v_1, v_2)| + |\pi(v_2, v_1)| = 2|\pi(v_1, v_2)|$.

Assume now that $h \geq 3$. The algorithm examines all possible choices of the lowest (minimizing the y-coordinate) vertex of C. Let v_1 be such a lowest vertex, let l_1 be the horizontal line through v_1, and let h_1 be the horizontal half-plane above l_1. Since no line in $\mathcal{L}$ is horizontal, $l_1 \notin \mathcal{L}$ and all lines in $\mathcal{L}$ intersect the half-plane h_1. Next, let $m = |V(\mathcal{A}(\mathcal{L})) \cap h_1| + 1$, and set $v_m = v_1$. Let $V_1 = (V(\mathcal{A}(\mathcal{L})) \cap h_1) \cup \{v_m\}$, and let the vertices $v_2, \ldots, v_{m-1} \in V_1$ be ordered according to increasing angle with respect to the leftwards (horizontal) ray from v_1 (in clockwise order around v_1). In case of ties, we order vertices by increasing distance from v_1. For this ordering scheme, if $C = (v_1, v_{i_2}, v_{i_3}, \ldots, v_m)$ is a solution to MCH, we have $1 < i_2 < i_3 < \cdots < m$. Consequently, only ordered pairs (v_j, v_k) of vertices, where $1 \leq j < k \leq m$ and either $j \neq 1$ or $k \neq m$, are candidate hull edges for this solution; furthermore, for $1 < j < k < m$, we can restrict ourselves to pairs (v_j, v_k) for which v_j and v_k are not collinear with $v_1 = v_m$.

For $1 \leq j < k \leq m$ and either $j \neq 1$ or $k \neq m$, let $l_{j,k}$ denote the oriented line through v_j and v_k (oriented from v_j to v_k), let $h_{j,k}$ denote the (closed) half-plane on the right of $l_{j,k}$, and let $C_{j,k} = h_1 \cap h_{j,k}$ denote the cone that is the intersection of h_1 and $h_{j,k}$. Finally, let $R_{1,k} = \{v_1\}$ for $1 < k \leq m$, and let $R_{j,k}$, for $1 < j < k \leq m$, denote the (possibly unbounded) closed triangular region $C_{j,k} \cap h_{j,m}$; see Fig. 3.

Depending on whether the slope of v_jv_k is positive or negative, the apex of the cone $C_{j,k}$ will lie to the left or to the right, respectively, of v_1; the figure illustrates both cases. Clearly, both $C_{j,k}$ and $R_{j,k}$ depend also on the choice of v_1; however, for notational convenience, we omit showing the explicit dependence — and for the remainder of our algorithm description, we fix a particular choice of v_1; the outer loop of the algorithm iterates over all $O(n^2)$ choices of v_1.

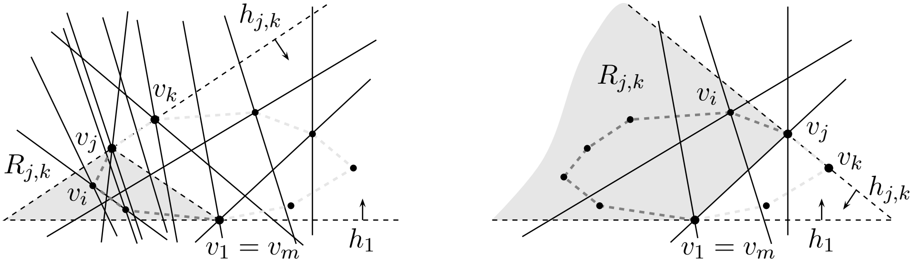


Fig. 3. SubProblem(v_1, v_j, v_k), $1 \leq j < k \leq m$, either $j \neq 1$ or $k \neq m$; $v_i \in R_{j,k}$

We say that an (ordered) pair (v_j, v_k) of vertices in V_1, for $1 \leq j < k \leq m$ and either $j \neq 1$ or $k \neq m$, is *eligible* if every line $\ell \in \mathcal{L}$ intersects the cone $C_{j,k}$. If (v_j, v_k) is not eligible, it does not need to be considered as a candidate edge of the convex polygon Q that is the desired solution to the MCH problem: there is some line in $\mathcal{L}$ that does not intersect the cone $C_{j,k}$, and therefore does not intersect any convex polygon Q having the edge v_jv_k, since $Q \subset C_{j,k}$.

For each eligible pair (v_j, v_k), SubProblem(v_1, v_j, v_k) is defined as follows (recall that $1 \leq j < k \leq m$ and either $j \neq 1$ or $k \neq m$).

SubProblem(v_1, v_j, v_k): Compute a minimum-length convex (right-turning) chain from v_1 to v_j such that the chain lies within the region $R_{j,k}$ and it intersects every line $\ell \in \mathcal{L}_{j,k}$, where $\mathcal{L}_{j,k}$ denotes the subset of lines $\ell \in \mathcal{L}$ that intersect $R_{j,k}$ but do not intersect the (closed) line segment v_jv_m. (The lines in $\mathcal{L}_{j,k}$ are the responsibility of the subproblem to visit; recall that $v_m = v_1$ and notice that $\mathcal{L}_{1,k} = \emptyset$.)

Next, for an eligible pair (v_j, v_k), let $f(j,k)$ denote the minimum length of a chain from v_1 to v_j that solves SubProblem(v_1, v_j, v_k); if (v_j, v_k) is not eligible, then we set $f(j,k) = \infty$. Note that $f(1,k) = 0$ if (v_1, v_k) is eligible. Our overall problem is to find an eligible pair (v_j, v_m) such that $f(j,m) + |\pi(v_j, v_m)|$ is minimized over all eligible pairs (v_j, v_m); in such a case, $v_jv_m = v_jv_1$ is the last edge of the convex polygon Q formed by an optimal cyclic sequence, starting at v_1 and going in the clockwise manner around Q, returning to $v_m = v_1$.

The dynamic programming recursion (Bellman equation) is defined as follows. The base of the recursion is $f(1,k) = 0$, if (v_1, v_k) is eligible, and $f(1,k) = \infty$, if (v_1, v_k) is not eligible. Next, for an eligible pair (v_j, v_k), $1 < j < k \leq m$,

$$f(j,k) = \min_{i \in I_{j,k}} (f(i,j) + |\pi(v_i, v_j)|),$$

where $I_{j,k}$ is the set of all indices i with $1 \leq i < j$ such that (i) $v_i \in R_{j,k}$ (which enforces convexity of the chain), (ii) (v_i, v_j) is eligible, and (iii) each line $\ell \in \mathcal{L}_{j,k} \setminus \mathcal{L}_{i,j}$ intersects the segment $v_i v_j$. In particular, note that $f(j,k) = |\pi(v_1, v_j)|$ if $\mathcal{L}_{j,k} = \emptyset$ (and (v_1, v_j) is eligible), so that a one-edge chain from v_1 to v_j suffices to meet all lines that are the responsibility of the subproblem. And, if $I_{j,k} = \emptyset$, then $f(j,k) = \infty$; recall too that we defined $f(j,k) = \infty$, if (v_j, v_k) is not eligible.

We tabulate the values $f(i,j)$ in the clockwise angular order around v_1, so that the values $f(i,j)$, for $1 \leq i < j < k \leq m$, are known by the time they are needed to compute $f(j,k)$.

Testing Conditions (i)-(iii). Clearly, condition (i) for our Bellman recursion can be easily checked in $O(1)$ time per candidate v_i. Next, condition (ii) can be also checked in $O(1)$ time by first pre-computing and tabulating eligibility for all pairs of vertices (and a fixed choice of v_1); for each of the $O(n^6)$ choices of v_1, v_j, v_k, eligibility of (v_j, v_k) is determined in time $O(n)$ time by testing each of n lines in $\mathcal{L}$ for intersection with $C_{j,k}$.

For efficient testing condition (iii) — in $O(n^2)$ total time ($O(1)$ time per candidate v_i) — some additional preprocessing must be done (details omitted).

Correctness of the Approach. The correctness proof is based upon the following claim.

Claim. *The values of $f(j,k)$ tabulated in solving the recursion are the lengths of optimal solutions to the corresponding subproblems SubProblem(v_1, v_j, v_k).*

This claim follows by induction on the index $j < k$, for any primarily fixed $1 < k \leq m$. Namely, the claim holds trivially for $j = 1$, by our definition of $f(1,k)$. Next, the induction hypothesis is: assume that $f(j',k')$ is the minimum length of a chain solving SubProblem$(v_1, v_{j'}, v_{k'})$ for all values of $1 \leq j' < k' < k$, having a finite-length solution. Consider now the SubProblem(v_1, v_{j+1}, v_k), for $j+1 < k$, and let $(v_1, v_{j_1}, v_{j_2}, \ldots, v_{j_N}, v_{j+1})$ be an optimal (minimum-finite-length) chain solving the subproblem. In the recursion, we must have considered the choice $v_i = v_{j_N}$, $j_N \leq j$, the next-to-last vertex on the optimal chain, and v_i must have satisfied conditions (i-iii) in our Bellman recursion. The subchain $(v_1, v_{j_1}, v_{j_2}, \ldots, v_{j_N})$ must be optimal and finite-length for the SubProblem(v_1, v_{j_N}, v_{j+1}), and so $f(j+1,k) = f(j_N, j+1) + |\pi(v_{j_N}, v_{j+1})|$ — otherwise, if a shorter feasible chain existed, we could improve upon the chain $(v_1, v_{j_1}, v_{j_2}, \ldots, v_{j_N}, v_{j+1})$. Consequently, since $1 \leq j_N < j+1 < k$, the value $f(v_{j_N}, v_j)$ for SubProblem(v_1, v_{j_N}, v_{j+1}) has been correctly computed by the induction hypothesis, and thus the hypothesis continues to hold, and so the claim. Thereby we have the following theorem.

Theorem 1. *The watchman route problem for n lines in the plane can be solved in $O(n^8)$ time.*

Proof. We have already given the algorithm and proved correctness above. It remains to argue that the algorithm can be implemented to run in time $O(n^8)$.

For each of the $O(n^2)$ choices of v_1, we sort the vertices in $O(n^2 \log n)$ time, and then we evaluate $f(j,k)$ for $O(n^4)$ pairs (v_j, v_k). Each evaluation requires consideration of $O(n^2)$ choices of v_i in evaluating the recursion, with conditions (i)-(iii) determining $I_{j,k}$ checkable in time $O(1)$ per v_i, after $O(n^7)$-time preprocessing for computing the lengths of all paths $\pi(v_i, v_j)$, determining eligible pairs, and some additional data (details omitted), yielding overall time $O(n^8)$. □

Half-Lines in the Plane. A natural question arises whether the above approach can be extended to the watchman route problem for half-lines in the plane. And, an example depicted in Fig. 4 shows that we cannot simply reduce this watchman route variation to the problem of determining the minimum-length cyclic sequence $(v_1, \dots, v_h, v_1)$ such that the convex polygon $v_1, \dots, v_h$ intersects all half-lines (and then to concatenate the shortest paths $\pi(v_1, v_2), \dots, \pi(v_{h-1}, v_h)$, and $\pi(v_h, v_1)$): in this example, while the line segment v_1v_2 intersects all of the half-lines, the shortest tour $\pi(v_1, v_2) \cup \pi(v_2, v_1)$ visiting v_1 and v_2 misses the half-line h.

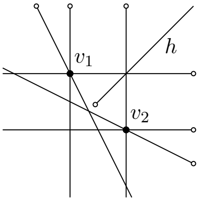


Fig. 4. The line segment v_1v_2 intersects all half-lines, but the shortest tour $\pi(v_1, v_2) \cup \pi(v_2, v_1)$ visiting v_1 and v_2 misses the half-line h. The endpoints of the half-lines are marked with small empty circles.

2.2 The Watchman Route Problem for Lines in 3D

We relate our watchmen route problem to the Geometric Traveling Salesman Problem (GTSP) [10,18], which can be formulated as the following decision problem: *Given a set of n lattice points in the plane, and a positive integer m, does there exists a tour of total length at most m that visits all the points?* GTSP is known to be NP-hard with respect to both the L_1 and the L_2 metric [9,18], and based upon this result, we obtain the following.

Theorem 2. *The watchman route problem for lines (or line segments) in 3D is NP-hard. The problem remains so even for orthogonal lines (or line segments).*

In the next section we consider the watchman route problem for line segments in the plane. We provide a polylogarithmic approximation algorithm for the WRS problem, running in polynomial time. The algorithm also applies to the watchman route problem for lines in 3D.

3 Segments in the Plane

In this section, we discuss the watchman route problem for line segments in the plane. In particular, we show the WRS problem to be NP-hard even if segments are restricted to be axis-aligned (with a simpler proof than [26]). Then we provide an approximation algorithm with ratio $O(\log^3 n)$ for the case of any connected set of segments in any dimension, and show that an approximation algorithm with a constant ratio exists for certain special cases of the watchman route problem for segments. Next, we extend our dynamic programming approach for the watchman route problem for lines in the plane to compute an optimal watchman route for simple connected sets of line segments in the plane. Finally, we provide a faster exact algorithm for the WRS problem in almost simple grids.

3.1 NP-Hardness

Theorem 3. *The watchman tour problem for segments in the plane is NP-hard. The problem remains so even for axis-aligned line segments.*

Proof. We adapt the NP-hardness proof of computing a shortest watchman tour in a polygon with holes [7]; we omit the details.

3.2 Approximation Algorithm

We apply the polylogarithmic approximation algorithm for the *group (or "one-of-a-set") Steiner tree problem* [15]: Given an undirected graph $G = (V, E)$ with weighted edges and n vertices, and given k subsets of V (called *groups* of vertices), find a minimum-weight tree that has at least one vertex from each of the groups. Fakcharoenphol et al. [8] gave an approximation algorithm with ratio $O(\log^2 n \log k)$ for general graphs, and to apply this approximation result to our problem, the graph $G = (V, E)$ is set to the weighted planar graph $G(\mathcal{S})$ (see Section 1), and groups of vertices correspond to the sets of intersection points along each of the n input segments; thus, there are $k = n$ groups. Observe that any watchman route contains a group Steiner tree of lesser weight; conversely, any group Steiner tree yields (by doubling) a watchman route for the segments in $\mathcal{S}$. So our algorithm finds a group Steiner tree of small weight in the planar graph G with respect to the above groups. By doubling the edges of the resulting tree, a watchman route for $\mathcal{S}$ is obtained. Hence, the approximation algorithm of Fakcharoenphol et al. [8] yields an approximation ratio $O(\log^2 |V| \log k) = O(\log^3 n)$; we point out that it applies to any dimension.

Theorem 4. *There is an approximation algorithm with ratio $O(\log^3 n)$ for computing a shortest watchman route for a connected set of n line segments in $\mathbb{R}^d$, for any dimension d.*

3.3 Light Segments

In the special case that each line segment $s \in \mathcal{S}$ has at most a constant number of intersection points (with other line segments) on s, we apply the result of Slavik [20]: the group TSP (what Slavik called the "errand scheduling problem") has a $\frac{3c}{2}$-approximation algorithm, where c is the maximum size of a group. (Notice that we do not require that at most a constant number of line segments may have a point in common).

Theorem 5. *Suppose that for each line segment $s \in \mathcal{S}$, there are at most c intersection points on s. Then, there is a polynomial time algorithm for the watchman route problem with an approximation factor of $\frac{3c}{2}$.*

3.4 Simple Arrangements

An arrangement $\mathcal{A}(\mathcal{S})$ of line segments is called *simple* if all the endpoints of the segments in $\mathcal{S}$ lie on the outer face of the arrangement and there exists $\varepsilon > 0$ such that if each segment is extended by ϵ in both directions, its new endpoints still lie on the outer face. For example, the arrangement shown in Fig. 5(a) is simple, while the arrangement shown in Fig. 5(b) is not. Guarding problems on simple arrangements restricted to axis-aligned segments, called *grids*, has been studied in [11,13].

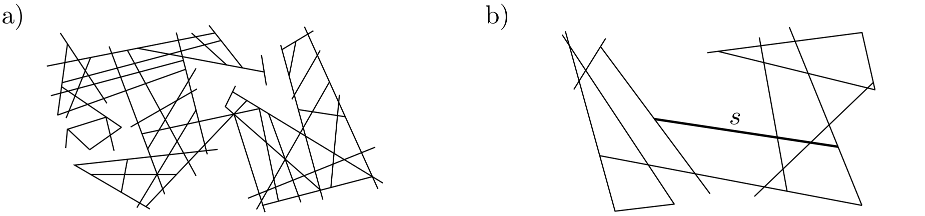


Fig. 5. (a) A simple arrangement. (b) An arrangement that is not simple, since the segment s cannot be extended without losing the property that its endpoints lie on the outer face.

An arrangement $\mathcal{A}(\mathcal{L})$ of lines can be thought of as the simple arrangement of line segments that results from cutting off all the crossings of $\mathcal{A}(\mathcal{L})$ (and the respective subsegments of lines) by a large enough disk. Our dynamic programming extends to the case of simple arrangements of line segments (of arbitrary orientations).

Theorem 6. *The watchman route problem for a simple arrangement of n line segments in the plane can be solved in polynomial time.*

3.5 Almost Simple Grids

A grid is *almost simple* if for every grid segment at least one of its endpoints lies on the outer face of the planar subdivision formed by the grid. Clearly, a simple grid is almost simple.

Restricting segments to be axis-aligned allows a faster algorithm that computes an optimal watchman route for almost simple grids. Namely, we obtain the following.

Theorem 7. *The watchman route problem for an almost-simple grid with n segments can be solved in $O(n \log n)$ time.*

4 Concluding Remarks

We conclude with a few open problems concerning watchman routes in line/half-line/segment arrangements:

(i) Can the running time of our algorithm for the watchman route problem for lines in the plane be substantially improved?
(ii) What is the complexity of the watchman route problem for half-lines (rays) in the plane?
(iii) Can the $O(\log^3 n)$ approximation factor for segments in the plane be reduced?
(iv) What is the complexity of the watchman route problem for an arrangement of planes in 3D?

Acknowledgment. The authors thank Marc Benkert, Gruia Călinescu, Till Nierhoff, Étienne Schramm, Micha Sharir, and Alexander Wolff for valuable remarks and interesting discussions on the topic. Dumitrescu was partially supported by NSF CAREER grant CCF-0444188 and NSF grant DMS-1001667. Mitchell was partially supported by NSF (CCF-1018388) and by Metron Aviation (subcontract from NASA Ames). Part of Żyliński's research was done at University of Wisconsin–Milwaukee, with support from NSF (CCF-0444188).

References

1. Brimkov, V.E., Leach, A., Mastroianni, M., Wu, J.: Guarding a set of line segments in the plane. Theoretical Computer Science 412(15), 1313–1324 (2011)
2. Chin, W., Ntafos, S.: Optimum watchman routes. In: Proc. 2nd Symposium on Computational Geometry, pp. 24–33 (1986)
3. Chin, W., Ntafos, S.: Optimum watchman routes. Information Processing Letters 28(1), 39–44 (1988)
4. Chvátal, V.: A combinatorial theorem in plane geometry. Journal of Combinatorial Theory, Series B 18, 39–41 (1997)
5. Dror, M., Efrat, A., Lubiw, A., Mitchell, J.S.B.: Touring a sequence of polygons. In: Proc. 35th Symposium on Theory of Computing, pp. 473–482 (2003)

6. Dumitrescu, A., Mitchell, J.S.B.: Approximation algorithms for TSP with neighborhoods in the plane. Journal of Algorithms 48(1), 135–159 (2003)
7. Dumitrescu, A., Tóth, C.D.: Watchman tours for polygons with holes. Computational Geometry: Theory and Applications 45(7), 326–333 (2012)
8. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: Proc. 35th ACM Symposium on Theory of Computing, pp. 448–455 (2003)
9. Garey, M.R., Graham, R., Johnson, D.S.: Some NP-complete geometric problems. In: Proc. 8th ACM Symposium on Theory of Computing, pp. 10–22 (1976)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1979)
11. Gewali, L.P., Ntafos, S.: Covering grids and orthogonal polygons with periscope guards. Computational Geometry: Theory and Applications 2(6), 309–334 (1993)
12. Hoffmann, F.: Private communication. In: The European Workshop on Computational Geometry (EuroCG 2000), Eilat, Israel (2000)
13. Kosowski, A., Małafiejski, M., Żyliński, P.: Cooperative mobile guards in grids. Computational Geometry: Theory and Applications 37(2), 59–71 (2007)
14. Mata, C.S., Mitchell, J.S.B.: Approximation algorithms for geometric tour and network design problems. In: Proc. 11th Symposium on Computational Geometry, pp. 360–369 (1995)
15. Mitchell, J.S.B.: Geometric shortest paths and network optimization. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 633–701. Elsevier (2000)
16. Ntafos, S.: On gallery watchmen in grids. Information Processing Letters 23, 99–102 (1986)
17. O'Rourke, J.: Art Gallery Theorems and Algorithms. Oxford University Press, New York (1987)
18. Papadimitriou, C.H.: Euclidean TSP is NP-complete. Theoretical Computer Science 4, 237–244 (1977)
19. Shermer, T.: Recent results in Art Galleries. Proc. of the IEEE 80, 1384–1399 (1992)
20. Slavik, P.: The errand scheduling problem, CSE Technical Report 97-02, University of Buffalo (1997)
21. Tan, X.: Fast computation of shortest watchman routes in simple polygons. Information Processing Letters 77(1), 27–33 (2001)
22. Tan, X.: A linear-time 2-approximation algorithm for the watchman route problem for simple polygons. Theoretical Computer Science 384(1), 92–103 (2007)
23. Tóth, C.D.: Illumination in the presence of opaque line segments in the plane. Computational Geometry: Theory and Applications 21(3), 193–204 (2002)
24. Tóth, C.D.: Illuminating disjoint line segments in the plane. Discrete & Computational Geometry 30(3), 489–505 (2003)
25. Urrutia, J.: Art gallery and illumination problems. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 973–1027. Elsevier (2000)
26. Xu, N., Brass, P.: On the complexity of guarding problems on orthogonal arrangements. In: Abstracts of the 20th Fall Workshop on Computational Geometry (FWCG 2010), #33 (2010)

Kinetic Pie Delaunay Graph and Its Applications

Mohammad Ali Abam[1,2], Zahed Rahmati[3], and Alireza Zarei[4]

[1] Dept. of Computer Engineering, Sharif University of Technology, Tehran, Iran
abam@sharif.edu

[2] Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
abam@ipm.ir

[3] Dept. of Computer Science, University of Victoria, Victoria, BC, Canada
rahmati@uvic.ca

[4] Dept. of Mathematical Science, Sharif University of Technology, Tehran, Iran
zarei@sharif.edu

Abstract. We construct a new proximity graph, called the *Pie Delaunay graph*, on a set of n points which is a super graph of *Yao graph* and *Euclidean minimum spanning tree* (*EMST*). We efficiently maintain the *Pie Delaunay graph* where the points are moving in the plane. We use the kinetic *Pie Delaunay graph* to create a kinetic data structure (*KDS*) for maintenance of the *Yao graph* and the *EMST* on a set of n moving points in 2-dimensional space. Assuming x and y coordinates of the points are defined by algebraic functions of at most degree s, the structure uses $O(n)$ space, $O(n \log n)$ preprocessing time, and processes $O(n^2 \lambda_{2s+2}(n) \beta_{s+2}(n))$ events for the *Yao graph* and $O(n^2 \lambda_{2s+2}(n))$ events for the *EMST*, each in $O(\log^2 n)$ time. Here, $\lambda_s(n) = n\beta_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols. Our *KDS* processes nearly cubic events for the *EMST* which improves the previous bound $O(n^4)$ by Rahmati *et al.* [1].

Keywords: Euclidean minimum spanning tree, Yao graph, Pie Delaunay triangulation, kinetic data structures.

1 Introduction

Investigating geometric problems on moving points, known as kinetic geometric problems, has been studied extensively in the past decade [2, 3, 4, 5]. In this setting, the points are moving in the plane and our goal is to show a data structure maintaining the combinatorial structure of a special attribute during the motion. We assume that the trajectory of the point p_i at time t ($p_i(t)$) is defined by two polynomial functions of maximum degree s for x and y coordinates of p_i ($p_i(t) = (x_i(t), y_i(t))$).

In this paper, we present a simple kinetic data structure for maintenance of the following proximity problems. In Euclidean space, for a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points, there exists the complete graph $G(V, E)$ where $V = P$ and E is the

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 48–58, 2012.

set of edges in the graph such that the weight of each edge is the Euclidean distance between its two endpoints. An *Euclidean minimum spanning tree* (*EMST*) of G is a connected sub-graph of G where the sum of the weights of its edges is the minimum possible. The *Yao graph* [6] of P can be constructed by partitioning the plane, for each point, into k wedges with equal angles $2\pi/k$ and connecting each point to the closest point in each of its k wedges. In the rest of the paper when we talk about the *Yao graph* it means that $k = 6$.

We present a *KDS* for the *Yao graph* and a new *KDS* for the *EMST* which is an improvement of the previous *EMST KDS* by Rahmati *et al.* [1] (our *KDS* processes nearly cubic events but the *KDS* in [1] processes $O(n^4)$ events). Guibas *et al.* [7] presented a *KDS* for Delaunay triangulation based on a *circle*, and Abam *et al.* [2] presented a *KDS* based on a *diamond*. There we partition a disk into six wedges with equal angles which creates six convex shapes; a Delaunay triangulation is then constructed based on each of these wedges. The union of all these triangulations is a sparse proximity graph. This new proximity graph, which we call *Pie Delaunay graph*, is a super graph of the *Yao graph* and the *EMST*.

Notation. $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols. Intuitively, if we have a set of n moving points where the trajectory of each point is an algebraic function with at most degree s then the number of changes for the lowest point along the y-axis is $\lambda_s(n)$. Here, $\beta_s(n) = \frac{\lambda_s(n)}{n}$ and $\alpha(n)$ is the Inverse Ackermann function.

Related work. Fu and Lee (1991) [8] proposed the first algorithm for maintenance the *EMST* on a set of moving points. The algorithm uses $O(sn^4 \log n)$ preprocessing time where s is the maximum degree of the algebraic functions defining the trajectory of the points and uses $O(m)$ space where m is the maximum number of the changes of the *EMST* from time $t = 0$ to $t = \infty$. At any given time, the algorithm constructs the *EMST* in linear time. Agarwal *et al.*(1998) [9] proposed an algorithm for a restricted kinetic version of the *EMST* over a graph where the distance between each pair of points in the graph is defined by linear function of time. Processing time of the algorithm for each combinatorial change of the *EMST* is $O(n^{\frac{1}{2}} \log^{\frac{3}{2}} n)$.

The kinetic data structure (*KDS*) framework was introduced by Basch (1999) [10]. To maintain a special attribute of a set of moving points, a *KDS* defines a set of *certificates* which certify the correctness of the attribute. During the time when a certificate fails, one must update the value of the attribute and then, build the new set of certificates to satisfy the correctness of the attribute. Therefore, it suffices to compute the failure times of these certificates, *events*, and put them in an event queue. Whenever the time of the next event in the queue is equal to the current time, one invokes a repair mechanism to update the value of the attribute and replace the failed certificate(s) with new valid one(s). The set of data structures and the update mechanism used to update these certificates and maintain the attributes is called a *KDS*. The kinetic data structure framework has been used for solving many of the geometric problems in kinetic environments.

Basch *et al.* (1999)[11] presented an approximation algorithm for $(1+\epsilon)$-*EMST*. Their *KDS* uses $O(\epsilon^{\frac{-(d-1)}{2}} n \log^{d-1} n)$ space, $O(\epsilon^{\frac{-(d-1)}{2}} n \log^{d-1} n)$ preprocessing time, and processes $O(\epsilon^{-(d-1)} n^3)$ events, each in $O(\log^d n)$ time where d is the dimension of the points. Recently, Rahmati *et al.* (2011)[1] improved the previous result by Fu and Lee [8]. They presented the exact *KDS* for maintenance of the *EMST* on a set of n moving points in 2-dimensional space. They build a *KDS* of space $O(n)$ in $O(n \log n)$ preprocessing time and their *KDS* processes $O(n^4)$ events, each in $O(\log^2 n)$ time.

Our Results. We introduce the *Pie Delaunay graph* which is a super graph of *Yao graph*. Since the set of *EMST* edges is a subset of the set of *Yao graph* edges the *Pie Delaunay graph* includes the *EMST*. We maintain the *Pie Delaunay graph* which enables us to maintain *Yao graph* and *EMST*. Our *KDS* uses $O(n)$ space, $O(n \log n)$ preprocessing time, and processes $O(n^2 \lambda_s(n) \beta_{s+2}(n))$ events for the *Yao graph* and $O(n^2 \lambda_{2s+2}(n))$ events for the *EMST*, each in $O(\log^2 n)$ time.

We describe our *KDS* in two sections. Section 2 contains the construction of the *Pie Delaunay graph* and its *KDS*; the *KDS* in this section maintains the *Pie Delaunay graph* during the motion. Next, we show the application of the *Pie Delaunay graph* in Section 3: first we construct the *Yao graph* and the *EMST* and then in Subsection 3.2, we maintain the *Yao graph* based on the kinetic *Pie Delaunay graph* and in Subsection 3.3, we present the kinetic *EMST* based on the kinetic *Yao graph*.

2 Pie Delaunay Graph

In this section, we summarize the construction of the *Pie Delaunay graph* and then we present a *KDS* for it.

2.1 The Construction of the Pie Delaunay Graph

Partition a disk into six wedges $\sigma_0, ..., \sigma_5$, each of angle $\pi/3$ and the origin as their common apex, where σ_i spans the orientation $[i\pi/3, (i+1)\pi/3]$, and call any translated and scaled copy of σ_i an *i-pie*—see Fig. 1. Let P be a set of points in the plane. We denote the constructed Delaunay triangulation of the point set

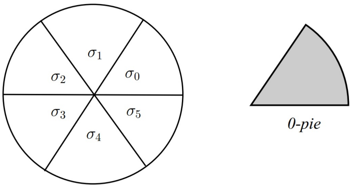


Fig. 1. Partitioning a circle into six pies and the 0-pie

P based on σ_i by $\mathcal{DT}_i(P)$ and define it as follow: For two points $p, q \in P$, the edge pq is an edge in $\mathcal{DT}_i(P)$ if and only if there is an i-pie where p and q are on its boundary and it does not contain any other points from P. Fig. 2 shows $\mathcal{DT}_0(P)$. In this figure, pq, qr, rp and s_1s_2 are the edges of $\mathcal{DT}_0(P)$ because for each of these edges there is a *0-pie* which dose not contain any other point from P. The *Pie Delaunay graph* ($\mathcal{DG}(P)$) is the union of all $\mathcal{DT}_i(P)$ for $i = 0, ..., 5$. We denote the set of $\mathcal{DG}(P)$ edges by $\mathcal{E}(\mathcal{DG}(P))$. Then, pq is an edge of $\mathcal{DG}(P)$ if and only if it is an edge in $\mathcal{DT}_i(P)$ where $0 \leq i \leq 5$. Each wedge σ_i is a convex shape and so, using the approach of [12], its corresponding Delaunay triangulation $\mathcal{DT}_i(P)$ can be constructed (based on σ_i) in $O(n \log n)$ time. Therefore, we can construct the *Pie Delaunay graph* in $O(n \log n)$ time.

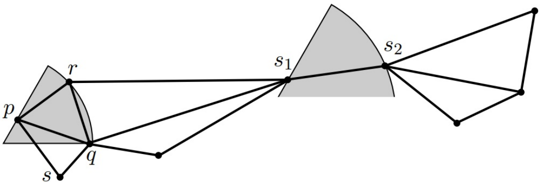


Fig. 2. Delaunay Triangulation based on 0-pie

2.2 Kinetic Pie Delaunay Graph

Given $\mathcal{E}(\mathcal{DG}(P))$, we show how to maintain the *Pie Delaunay graph* with processing time $O(\log n)$ for each certificate failure.

In our *KDS* we define two certificates *NotInPie* and *NotInCone*. Call the edges on the boundary of $\mathcal{DT}_i(P)$ *i-hull* edges. Every interior edge pq, which is not an i-hull edge, is incident to two triangles. Call the two triangles a quadrilateral, and let r and s be the two other vertices of the quadrilateral. For the convex shape i-pie that passes through p, q, and r, we have a *NotInPie* certificate which certifies that point s is outside of the pie, as shown in Fig. 2. When the certificate fails, we replace pq by rs. In general, when a corresponding certificate of each quadrilateral fails, we perform an edge flip.

Each i-pie has three edges. By removing one of the edges and extending the other two edges a cone can be created, call these cones i-cones, see Fig. 3(a). An edge pq is an i-hull edge if and only if there exists an i-cone such that p and q are on its boundary and the i-cone does not contains any other points—see the edge pq in Fig. 3(b). Each edge pq of the i-hull is incident to at most four other i-hull edges, call them by ps_1, ps_2, qs_3, qs_4, and incident to at most one triangle. Let r be the third vertex of this triangle if it exists; r can be one of the s_i where $0 \leq i \leq 4$. For the i-cone passing through p and q, we maintain at most five *NotInCone* certificates certifying that r and s_i's are outside of the i-cone. Whenever a *NotInCone* certificate fails we either delete or insert a vertex into the i-hull and then, we delete or insert an edge into the triangulation.

Thus, when a *NotInCone* certificate or a *NotInPie* certificate fails we replace the invalid certificates with the new valid ones which causes a constant number

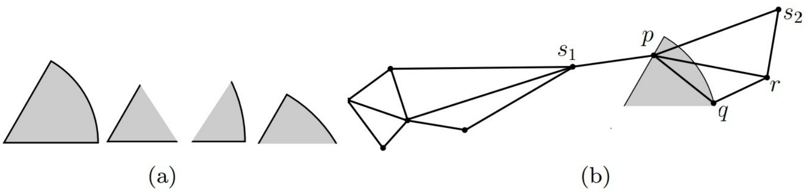


Fig. 3. (a) An i-pie and the three types of cones defined by it. (b) A hull edge pq corresponds to an i-cone.

of changes to the data structure, because the number of the invalid certificates is constant. After the updating, we also have to calculate the next failure times of the new valid certificates and place them in the queue which takes $O(\log n)$ time; the first element of the queue shows the next time that a certificate will be invalid. Thus, the following lemma results from the above discussion.

Lemma 1. *A change in the Pie Delaunay graph happens when a* NotInPie *certificate or a* NotInCone *certificate is invalid. The Pie Delaunay graph can be maintained kinetically in $O(\log n)$ time per event.*

Proof. Each edge of the *Pie Delaunay graph* is either an interior edge or an external (i-hull) edge. For the interior edge pq there exists an i-pie which p and q are on its boundary and it does not contain any other points. If we scale the i-pie such as p and q are on its boundary then a new point (r) will be incident to the boundary of the i-pie. In this case we need to define a certificate certifying p, q, and r are on the boundary of the i-pie and it does not contain any other points from P (*NotInPie* certificate). Similarly, for the external edge $p'q'$, we define a certificate certifying p' and q' are on the boundary of an i-cone and it does not contain any other points (*NotInCone* certificate). Thus, it satisfies our definition of the *Pie Delaunay graph* and for maintenance of the *Pie Delaunay graph* and we just need to define two certificates *NotInPie* and *NotInCone.*

When one of these events happens we apply a constant number of edge insertions and edge deletions into the *Pie Delaunay graph* and a constant number of changes in the event queue. So, we maintain the *Pie Delaunay graph* kinetically in $O(1)$ time per each of these events, plus $O(\log n)$ time to update the event queue. □

For a set of n points in the Euclidean plane, Guibas *et al.* [7] have shown that the number of the combinatorial changes in the Delaunay triangulation based on *circle* is $O(n^2\lambda_s(n))$. We have the following theorem about the number of the combinatorial changes of the $\mathcal{DG}(P)$ which is based on the i-pie.

Theorem 1. *The number of all changes (edge insertions and edge deletions) of the Pie Delaunay graph on a set of n moving points with trajectory of algebraic function with at most degree s is $O(n^2\lambda_{2s+2}(n))$.*

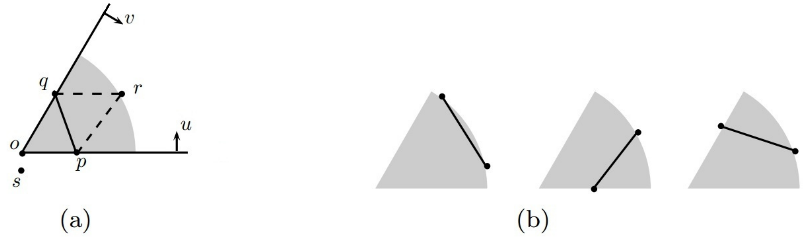
Proof. The number of convex-edge changes is $O(n^3)$ as three points are involved in any convex change. Since $n^3 = O(n^2\lambda_{2s+2}(n))$, we focus on the number of triangle changes in $\mathcal{DT}_i(P)$. For each edge pq of a triangle, four different cases are imaginable as shown in Fig. 4. It is easy to see for any triangle Δ, the case (a) of Fig. 4 happens to one of its edge. We charge any change to Δ to this edge. Therefore, we consider the number of the combinatorial changes of $\mathcal{DT}_i(P)$ for an arbitrary edge pq that satisfies case (a) of Fig. 4.

Two edges of an i-pie are line segments and one of them is an arc; call the line segments by ow_1 and ow_2. Let W_i be a wedge whose sides are created by removing the arc w_1w_2 of i-pie and extending the two line segments; the wedge W_i is the area between two half-lines $\overrightarrow{ow_1}$ and $\overrightarrow{ow_2}$. Let $\mathcal{V}(W_i)$ be the set of all points in the wedge W_i. In Fig. 4(a), a change for triangle pqr corresponding to pq happens when for some $t \in \mathcal{V}(W_i)$, the length of the edge ot becomes smaller than the length of the edge or.

Note that since the degree of each function describing each point's motion is at most s, each point of P except p and q, can be inserted inside the wedge W_i s times. Summing over all points in P there are $O(sn)$ insertion into $\mathcal{V}(W_i)$. The distance of these points from the apex o creates $O(sn)$ partial functions with at most degree $2s$. The number of the combinatorial changes corresponding to an arbitrary edge pq equals $\lambda_{2s+2}(sn)$ which is equal to the number of the breakpoints in the lower envelope of sn partial functions of at most degree $2s$ (Theorem 2.5. [13]). Since the maximum degree s is a constant, $\lambda_{2s+2}(sn) = O(\lambda_{2s+2}(n))$.

The number of all possible edges is $O(n^2)$ and therefore, the number of the combinatorial changes corresponding to all edges is $O(n^2\lambda_{2s+2}(n))$.

Besides the above changes for the edge pq, there exist other changes that happen when a point, such as s passes through the segment op or the segment oq and enters inside the area opq, see Fig. 4(a). Map each point $p = (x_p(t), y_p(t))$ to a point $p' = (u_p(t), v_p(t))$ in a new parametric plane where $u_p(t) = x_p(t) + \sqrt{3}y_p(t)$ and $v_p(t) = x_p(t) - \sqrt{3}y_p(t)$. Passing the point s through the segment op or the segment oq means that the point s' changes its u-coordinate or its v-coordinate with the u-coordinate or v-coordinate of p' or q', call these changes *swap-changes*. That is, the number of all swap-changes for all possibles is bounded with the number of all swaps between points in their ordering with respect to u-axis and v-axis. The number of the all u-swaps and v-swaps between points is $O(sn^2)$. □



(a) (b)

Fig. 4. Combinatorial changes for an arbitrary edge pq

From this discussion, Lemma 1, and the Theorem 1:

Theorem 2. *For a set of n points in the plane with the trajectories of algebraic functions with maximum degree s, the kinetic Pie Delaunay graph uses linear space and processes $O(n^2\lambda_{2s+2}(n))$ events, each in $O(\log n)$ time.*

3 The Applications

In this section we introduce new constructions for the *Yao graph* and the *EMST* and then, we consider the kinetic version of them.

3.1 The Constructions

For each point $p \in P$, partition the plane into k wedges $W_0(p), ..., W_{k-1}(p)$ of angle $2\pi/k$ where p is origin of the wedges and W_i spans the orientation $[2\pi i/k, 2\pi(i+1)/k]$. The *Yao graph* can be constructed by finding the closest point to p inside the wedge $W_i(p)$ where $0 \leq i \leq k-1$. For constructing the *EMST*, a version of the *Yao graph* where $k = 6$ is needed—we denote it by $\mathcal{YG}(P)$ and the set of its edges by $\mathcal{E}(\mathcal{YG}(P))$. The following lemma shows that the *Pie Delaunay graph* is a super graph of the *Yao graph*.

Lemma 2. $\mathcal{E}(\mathcal{YG}(P)) \subseteq \mathcal{E}(\mathcal{DG}(P))$.

Proof. Let W_i be a wedge whose sides are parallel to the sides of σ_i. For each point p, qp is an edge of $\mathcal{YG}(P)$ where q is the closest point to p inside $W_i(p)$, see Fig. 5. This means that, there is an i-pie where p and q are on its boundary and it dose not contain any other points of P. Therefore, $pq \in \mathcal{E}(\mathcal{DG}(P))$ and so, the *Pie Delaunay graph* includes the *Yao graph*. □

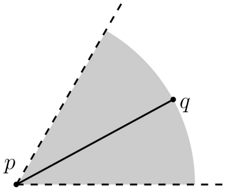


Fig. 5. An edge of a Yao graph is an edge of the Pie Delaunay Graph

Denote the *EMST* edges by $\mathcal{E}(EMST)$. In previous section, we noticed, using the approach of [12], $\mathcal{DG}(P)$ can be constructed in $O(n \log n)$ time. Cardinality of the $\mathcal{E}(\mathcal{DG}(P))$ is $O(n)$ and so, by a trace over the edges incident to each point of $\mathcal{DG}(P)$, we can construct the $\mathcal{YG}(P)$ in $O(n)$ time. $\mathcal{E}(EMST) \subseteq \mathcal{E}(\mathcal{YG}(P))$ [6] and since the number of edges in $\mathcal{E}(\mathcal{YG}(P))$ is linear, the *EMST* can be constructed in $O(n \log n)$ time using the Prim or Kruskal algorithm [14, 15] and so, the following lemma results.

Lemma 3. *Using linear space, the Yao graph and the EMST can be constructed in $O(n \log n)$ time.*

3.2 Kinetic Yao Graph

Now, assume the points start moving. To maintain $\mathcal{YG}(P)$ during the motion we introduce the *kinetic tournament tree* [4] which is a preliminary tool in the kinetic data structure framework.

Using a kinetic tournament tree, we can maintain the lowest point among a set of n moving points along the y-axis. The tournament tree on a set of n points is a balanced tree with the points stored at its leaves (in an arbitrary order). An internal node of the tournament tree maintains the lowest point between two children; the root of the tournament tree maintains the lowest point among all points. This tournament tree is known as *kinetic tournament tree* and the number of changes to the value at the root of the *kinetic tournament tree* is $\lambda_s(n)$ [3, 10]. We use a kind of tournament structure which supports insertions and deletions of points (*dynamic kinetic tournament tree*) [3]. The following theorem can be concluded from the Theorem 3.1. in [3] and it bounds the total number of events that may occur while inserting and deleting at most m points, at arbitrary locations, into a dynamic kinetic tournament.

Theorem 3. *A dynamic kinetic tournament, with a sequence of m insertions and deletions whose maximum size at any time is n (assuming $m \geq n$), generates at most $O(m\beta_{s+2}(n))$ events at the root. Processing an update or a tournament event takes $O(\log^2 n)$ worst-case time; the tournament on n elements can be constructed in $O(n)$ time.*

Let $\mathcal{E}(W_i(p))$ be the set of edges of the $\mathcal{DG}(P)$ inside the wedge $W_i(p)$ and incident to the point p. For each wedge $W_i(p)$, we have to maintain the closet point to p and so, corresponding to each $W_i(p)$ we construct a dynamic kinetic tournament $(DKT_i; i = 1, ..., 6n)$ whose elements are $\mathcal{E}(W_i(p))$. Therefore, at any time, the root of all dynamic kinetic tournaments are the edges of the *Yao graph* and so, the following theorem is resulted.

Theorem 4. *The KDS for maintenance of the Yao graph uses $O(n)$ space with preprocessing time $O(n \log n)$, and processes $O(n^2\lambda_{2s+2}(n)\beta_{s+2}(sn))$ events, each in $O(\log^2 n)$ time.*

Proof. We know that *Pie Delaunay graph* can be constructed in $O(n \log n)$ time and the cardinality of $\mathcal{E}(\mathcal{DG}(P))$ is $O(n)$. Each edge of the $\mathcal{DG}(P)$ is inserted into at most two of the DKT_i's which $i = 1, ..., 6n$. Let n_i be the number of elements in DKT_i. From Theorem 3, the construction time of DKT_i on $O(n_i)$ elements is $O(n_i)$ and so the construction time over all DKT_i's is $O(n)$. Thus, the *KDS* uses linear space with preprocessing time $O(n \log n)$.

Let m_i be the number of insertions/deletions into the DKT_i. From Theorem 1 we know that $\Sigma_{i=1}^{6n} m_i = O(n^2\lambda_{2s+2}(n))$. According to the Theorem 3, the number of all changes at the root of all DKT_i for $i = 1, ..., 6n$ is $\Sigma_{i=1}^{6n} O(m_i\beta_{s+2}(n)) = O(\beta_{s+2}(n)\Sigma_{i=1}^{6n} m_i) = O(n^2\lambda_{2s+2}(n)\beta_{s+2}(n))$ which each one can be handled in $O(\log^2 n)$ time. □

In our algorithm, we processed a nearly cubic number of events for maintenance of the *Yao graph* but the exact number of changes to the *Yao graph* is nearly square. For linearly moving points in the plane, Katoh *et al.* [16] showed the changes to the *Yao graph* is $O(n\lambda_4(n))$. In the following theorem we bound the number of the combinatorial changes of the *Yao graph* of a set of moving points with the trajectory of algebraic function of at most degree s.

Theorem 5. *The number of all changes in the Yao graph, when the points move with polynomial trajectory of at most degree s, is $O(n\lambda_{2s+2}(n))$.*

Proof. For an arbitrary point $p \in P$, each of other points of P can be inserted inside the wedge $W_i(p)$ s times and so, there exist $O(sn)$ insertion into the wedge $W_i(p)$. The distance of these points from p creates $O(sn)$ partial functions with at most degree $2s$; the lowest envelope of these partial functions corresponds to the closest point to p inside the wedge $W_i(p)$. The number of all changes in the lower envelope of sn partial functions with at most degree $2s$ corresponding to the point p is $\lambda_{2s+2}(n)$ (Theorem 2.5. [13]). Hence, the number of all changes to the *Yao graph* on a set of n moving points is $O(n\lambda_{2s+2}(n))$. □

3.3 Kinetic EMST

Our approach to maintain the *EMST* is based on the fact that the edges of the *EMST* are a subset of the edges of the *Yao graph*; A change in the combinatorial structure of the *EMST* depends on the orderings of the edge weights of the *Yao graph* edges.

Here, we maintain the edges of $\mathcal{YG}(P)$ (which are the root of DKT_i where $i = 1, ..., 6n$) in a sorted list ($L_{\mathcal{YG}}$) and whenever the ordering of two edges in this list is changed, we apply the required changes to the *EMST*. Therefore, we need to track these changes to update and maintain the *EMST* of a set of moving points. In particular, to maintain the *EMST* there exists two kinds of events that we should consider:

(a) edge insertion and edge deletion from $L_{\mathcal{YG}}$, and
(b) the change between two consecutive edges in $L_{\mathcal{YG}}$.

In the case (a), as soon as an edge is deleted from $L_{\mathcal{YG}}$ the new one is inserted; both of the deleted edge and the inserted edge are in the same dynamic kinetic tournament and have a common endpoint, call them by pq and pr. In this case, the deleted edge pq can be one of the *EMST* edges and so, we have to find a new edge reconstructing the *EMST*. It's easy to show that the new edge reconstructing the *EMST* is pr.

Now, we consider the case (b). Let $path(p_i, p_j)$ be the simple path between p_i and p_j in the *EMST* and $|e|$ be the Euclidean length of e. A change in $L_{\mathcal{YG}}$ corresponds to a pair of edges e and e' in $\mathcal{E}(\mathcal{YG}(P))$ where at time t^-, $|e| < |e'|$, and at time t^+, $|e| > |e'|$. Then, at time t, e may be replaced by e' in $\mathcal{E}(EMST)$. It is simple to prove the following lemma:

Lemma 4. *EMST changes if and only if at time t^-, $|e| < |e'|$, $e \in \mathcal{E}(EMST)$, $e' \notin \mathcal{E}(EMST)$, and $e \in path(p_i, p_j)$ where p_i and p_j are the end points of e' and at time t^+, $|e| > |e'|$.*

Such events can be detected and maintained within $O(\log n)$ time per operation using the link-cut tree data structure of [17].

Theorem 6 below bounds the number of the events of the *EMST* in our *KDS*.

Theorem 6. *The number of the combinatorial changes of the EMST in our KDS is* $O(n^2\lambda_{2s+2}(n))$.

Proof. The set of *Yao graph* edges is a superset of the set of the *EMST* edges and any change in the order of the consecutive edges in the sorted list of the *Yao graph* edges may change the *EMST*. Precisely, any change in the *Yao graph* causes insertion/deletion into the sorted list and each insertion may causes $O(n)$ changes in the *EMST*. The number of all insertions and deletions into the sorted list $L_{\mathcal{YG}}$ is $O(n\lambda_{2s+2}(n))$, see Theorem 5, and therefore, in our data structure, the number of the combinatorial changes of the *EMST* is $O(n^2\lambda_{2s+2}(n))$. □

We summarize all results of our *KDS* in the following theorem.

Theorem 7. *For a set of n moving points with polynomial trajectory of at most degree s, our KDS uses linear space and requires $O(n\log n)$ preprocessing time. The KDS processes $O(n^2\lambda_{2s+2}(n)\beta_{s+2}(n))$ events for the Yao graph and $O(n^2\lambda_{2s+2}(n))$ events for the EMST and each of these events can be handled in the worst case time $O(\log^2 n)$.*

4 Conclusion

In the paper, we presented the new proximity graph *Pie Delaunay graph* and then we maintained the kinetic data structure for the *Yao graph*. In our *KDS*, we process the number of nearly cubic events for the kinetic *Yao graph* but the exact number of the changes in the *Yao graph* is nearly square and so, finding a *KDS* which processes only nearly square events is a future direction.

For kinetic *EMST*, we handle a nearly cubic upper bound of topological changes but the tight upper bound is not known. For linearly moving points in the plane, Katoh *et al.* [16] proved an upper bound of $O(n^3 2^{\alpha(n)})$ for the number of the combinatorial changes of the *EMST* which was later improved to $O(n^{8/3}2^{\alpha(n)}\log^{4/3} n)$ by combining the results of Chan [18], and Marcus and Tardos [19]. Finding the tight upper bound for the combinatorial changes (events) of the *EMST* and finding a *KDS* for *EMST* processing the number of sub-cubic events are other future directions.

Acknowledgments. We thank Valerie King, Frank Ruskey, and Sue Whitesides for their useful conversations and comments.

References

[1] Rahmati, Z., Zarei, A.: Kinetic Euclidean Minimum Spanning Tree in the Plane. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2011. LNCS, vol. 7056, pp. 261–274. Springer, Heidelberg (2011)

[2] Abam, M.A., de Berg, M., Gudmundsson, J.: A simple and efficient kinetic spanner. Comput. Geom. Theory Appl. 43, 251–256 (2010)

[3] Alexandron, G., Kaplan, H., Sharir, M.: Kinetic and dynamic data structures for convex hulls and upper envelopes. Comput. Geom. Theory Appl. 36(2), 144–158 (2007)

[4] Basch, J., Guibas, L.J., Hershberger, J.: Data structures for mobile data. In: Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1997, pp. 747–756. Society for Industrial and Applied Mathematics, Philadelphia (1997)

[5] Kaplan, H., Rubin, N., Sharir, M.: A kinetic triangulation scheme for moving points in the plane. Comput. Geom. Theory Appl. 44(4), 191–205 (2011)

[6] Yao, A.C.C.: On constructing minimum spanning trees in k-dimensional spaces and related problems. SIAM J. Comput. 11(4), 721–736 (1982)

[7] Guibas, L.J., Mitchell, J.S.B.: Voronoi Diagrams of Moving Points in the Plane. In: Schmidt, G., Berghammer, R. (eds.) WG 1991. LNCS, vol. 570, pp. 113–125. Springer, Heidelberg (1992)

[8] Fu, J.J., Lee, R.C.T.: Minimum spanning trees of moving points in the plane. IEEE Trans. Comput. 40(1), 113–118 (1991)

[9] Agarwal, P.K., Eppstein, D., Guibas, L.J., Henzinger, M.R.: Parametric and kinetic minimum spanning trees. In: FOCS, pp. 596–605. IEEE Computer Society (1998)

[10] Basch, J.: Kinetic data structures. PhD Thesis, Stanford University (1999)

[11] Basch, J., Guibas, L.J., Zhang, L.: Proximity problems on moving points. In: Proceedings of the Thirteenth Annual Symposium on Computational Geometry, SCG 1997, pp. 344–351. ACM, New York (1997)

[12] Chew, L.P., Dyrsdale III, R.L.S.: Voronoi diagrams based on convex distance functions. In: Proceedings of the First Annual Symposium on Computational Geometry, SCG 1985, pp. 235–244. ACM, New York (1985)

[13] Agarwal, K.P., Sharir, M.: Davenport–schinzel sequences and their geometric applications. Technical report, Durham, NC, USA (1995)

[14] Kruskal, J.B.: On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society, 7 (1956)

[15] Prim, R.C.: Shortest connection networks and some generalizations. Bell Systems Technical Journal, 1389–1401 (November 1957)

[16] Katoh, N., Tokuyama, T., Iwano, K.: On minimum and maximum spanning trees of linearly moving points. In: Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, SFCS 1992, pp. 396–405. IEEE Computer Society, Washington, DC (1992)

[17] Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. 26(3), 362–391 (1983)

[18] Chan, T.M.: On levels in arrangements of curves. Discrete and Computational Geometry 29, 375–393 (2003)

[19] Marcus, A., Tardos, G.: Intersection reverse sequences and geometric applications. J. Comb. Theory Ser. A 113(4), 675–691 (2006)

Higher Order City Voronoi Diagrams

Andreas Gemsa[1], D.T. Lee[2,3], Chih-Hung Liu[1,2], and Dorothea Wagner[1]

[1] Karlsruhe Institute of Technology, Germany
[2] Academia Sinica, Taiwan
[3] National Chung Hsing University, Taiwan

Abstract. We investigate higher-order Voronoi diagrams in the *city metric*. This metric is induced by quickest paths in the L_1 metric in the presence of an accelerating transportation network of axis-parallel line segments. For the structural complexity of k^{th}-order city Voronoi diagrams of n point sites, we show an upper bound of $O(k(n-k)+kc)$ and a lower bound of $\Omega(n+kc)$, where c is the complexity of the transportation network. This is quite different from the bound $O(k(n-k))$ in the Euclidean metric [12]. For the special case where $k=n-1$ the complexity in the Euclidean metric is $O(n)$, while that in the city metric is $\Theta(nc)$. Furthermore, we develop an $O(k^2(n+c)\log(n+c))$-time iterative algorithm to compute the k^{th}-order city Voronoi diagram and an $O(nc\log^2(n+c)\log n)$-time divide-and-conquer algorithm to compute the farthest-site city Voronoi diagram.

1 Introduction

In many modern cities, e.g., Manhattan, the layout of the road network resembles a grid. Most roads are either horizontal or vertical, and thus pedestrians can move only either horizontally or vertically. Large, modern cities also have a public transportation network (e.g., bus and rail systems) to ensure easy and fast travel between two places. Traveling in such cities can be modeled well by the *city metric*. This metric is induced by quickest paths in the L_1 metric in the presence of an accelerating transportation network. We assume that the traveling speed on the transportation network is a given parameter $\nu > 1$. The speed while traveling off the network is 1. Further, we assume that the transportation network can be accessed at any point. Then the distance between two points is the minimum time required to travel between them.

For a given set S of n point sites (i.e., a set of n coordinates) and a transportation network in the plane, *the* k^{th}*-order city Voronoi diagram* $V_k(S)$ partitions the plane into *Voronoi regions* such that all points in a Voronoi region share the same k nearest sites with respect to the city metric.

The k^{th}-order city Voronoi diagram can be used to resolve the following situation: a pedestrian wants to know the k nearest facilities (e.g., k stores, or k hospitals) such that he can make a well-informed decision as to which facility to go to. For this kind of scenario, the k^{th}-order city Voronoi diagram provides a way to determine the k nearest facilities, by modeling the facilities as point sites.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 59–70, 2012.

The nearest-site (first-order) city Voronoi diagram has already been well-studied [1,4,6,11]. Its structural complexity (the size) has been proved to be $O(n + c)$ [1], where c is the complexity of the transportation network. Such a Voronoi diagram can be constructed in $O((n + c) \log(n + c))$ time [6]. However, to the best of our knowledge there is no existing work regarding k^{th}-order or farthest-site (i.e., $(n - 1)^{\text{st}}$-order) city Voronoi diagrams.

Contrary to k^{th}-order city Voronoi diagrams, k^{th}-order Euclidean Voronoi diagrams have been studied extensively for over thirty years. Lee [12] showed that the structural complexity of k^{th}-order Voronoi diagrams is $O(k(n - k))$. Lee also proved that the j^{th}-order Voronoi diagram can be constructed from the $(j-1)^{\text{st}}$-order Voronoi diagram, and developed an iterative construction method within $O(k^2 n \log n)$ time and $O(k^2 n)$ space. Chazelle and Edelsbrunner [7] made use of geometric duality and arrangements to develop a different algorithm which requires $O(n^2 \log n + k(n - k) \log^2 n)$ time and $O(k(n - k))$ space or $O(n^2 + k(n-k) \log^2 n)$ time and $O(n^2)$ space. Additionally, there are several randomized algorithms [2,14]. and on-line algorithms [3,5].

One of the most significant differences between the Euclidean metric and the city metric that influences the computation and complexity of Voronoi diagrams is the complexity of a bisector between two points. In the Euclidean or the L_1 metric such a bisector has constant complexity, while in the city metric the complexity may be $\Omega(c)$ [1] and can even be a closed curve. Since the properties of a bisector between two points significantly affect the properties of Voronoi diagrams, a k^{th}-order city Voronoi diagram can be very different from a Euclidean one. First, this property makes it non-trivial to apply existing approaches for constructing Euclidean Voronoi diagrams to the city Voronoi diagrams. Secondly, this property also indicates that the complexity of k^{th}-order Voronoi diagrams may depend significantly on the complexity of the transportation network.

In this paper, we derive bounds for the structural complexity of the k^{th}-order Voronoi diagram and develop algorithms for computing the k^{th}-order city Voronoi diagram. The remainder of this paper is organized as follows. In Section 2, we introduce two important concepts, wavefront propagation [1] and shortest path maps [6], which are essential for the proofs in the subsequent sections. In Section 3, we adopt the wavefront propagation to introduce a novel interpretation of the iterative construction method of Lee [12], and use this interpretation to derive an upper bound of $O(k(n - k) + kc)$ for the structural complexity of k^{th}-order city Voronoi diagrams, where c is the complexity of the transportation network. Then, we construct a worst-case example to obtain a lower bound of $\Omega(n + kc)$. Finally, we extend the insights of Section 3 to develop an iterative algorithm to compute k^{th}-order city Voronoi diagrams in $O(k^2(n + c) \log(n + c))$ time (see Section 4). Moreover, we give a divide-and-conquer approach to compute farthest-site city Voronoi diagrams in $O(nc \log n \log^2(n + c))$ time. We conclude the paper in Section 5.

Due to space constraints many proofs of this paper are missing, but can be found in the full version [10] of this paper.

2 Preliminaries

In this section we introduce the notation used throughout this paper for k^{th}-order city Voronoi diagrams. Then, we introduce two well-established concepts in the context of Voronoi diagrams, which are important for the proofs in the subsequent sections.

A transportation network is a planar straight-line graph $C = (V_C, E_C)$ with *isothetic* edges only, i.e., edges that are either horizontal or vertical, and all transportation edge have identical speed $\nu > 1$. We define $c := |V_C|$, and since the degree of a vertex in V_C is at most four, $|E_C|$ is $\Theta(c)$. We denote the distance of two points in the L_1 metric by d_1 and in the city metric by d_C. Similarly, we denote the bisector between two points by B_1 and B_C for the L_1 and city metric, respectively. Additionally, for the city metric we define the distance between a point $p \in \mathbb{R}^2$ and a set of points $H \subset \mathbb{R}^2$ to be $d_C(p, H) = \max_{q \in H} d_C(p, H)$. This allows us to define the bisector $B_C(H_1, H_2) = \{r \in \mathbb{R}^2 \mid d_C(r, H_1) = d_C(r, H_2)\}$ between two sets of points H_1 and H_2.

By $V_k(H, S)$ we denote a *Voronoi region* of $V_k(S)$ associated with a k-element subset $H \subset S$. The common boundary between two adjacent Voronoi regions $V_k(H_1, S)$ and $V_k(H_2, S)$ is called a *Voronoi edge*. This Voronoi edge is a part of $B_C(H_1, H_2) = B_C(p, q)$ where $H_1 \setminus H_2 = \{p\}$ and $H_2 \setminus H_1 = \{q\}$ [12]. The common intersection among more than two Voronoi regions is called a *Voronoi vertex*. Without loss of generality, we assume that no point in the plane is equidistant from four sites in S with respect to the city metric, ensuring that the degree of a Voronoi vertex is exactly three.

Wavefront Propagation. The wavefront propagation is a well-established model to define Voronoi diagrams [1]. In Section 3, we will use this concept to interpret the formation of $V_k(S)$ and analyze its structural complexity.

For a fixed site $p \in S$, let $W_p(x) = \{q \mid q \in \mathbb{R}^2, d_C(p, q) = x\}$. This means that for a fixed $x \in \mathbb{R}_0^+$ the wavefront $W_p(x)$ is the circle centered at p with radius x. We call p the *source* of $W_p(x)$. Note that we can view $W_p(x)$ as the wavefront at time x of the wave that originated in p at time 0. We refer to such a wavefront as W_p if the value of x is unimportant.

Initially, the wavefront W_p is a diamond. When it touches a part of the transportation network for the first time it changes its propagation speed and, hence its shape; see Fig. 1. Certain points on the transportation network play an important role to determine the structural complexity of k^{th}-order city Voronoi diagrams. Thus, we introduce the following definitions. For a point $v \in \mathbb{R}^2$, let $P(v)$ denote the *isothetic projection* of v onto the transportation network, i.e., we shoot an isothetic half-ray starting at v in each of the four directions and for each half-ray we add its first intersection with an edge of the transportation network to $P(v)$. It is easy to see that there are at most four such intersections. For a set $X \subset \mathbb{R}^2$, we denote the *isothetic projection of the set* X as $\mathcal{P}(X) = \bigcup_{v \in X} P(v)$. For a site $p \in S$, we call the set $A(p) = P(p) \cup V_C \cup \mathcal{P}(V_C) \cup \{p\}$ *activation points* (we added $\{p\}$ to the list for ease of argumentation in some of our proofs).

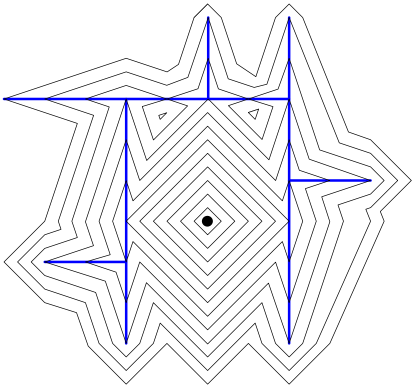


Fig. 1. Wavefront Propagation

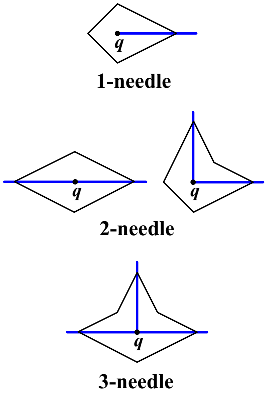


Fig. 2. 1-needle, 2-needle and 3-needle

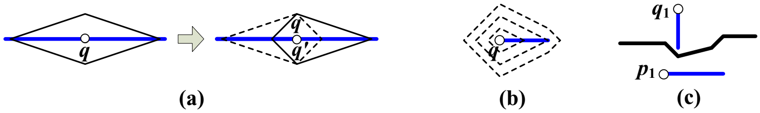


Fig. 3. (a) A 2-needle is two 1-needles. (b) Wavefront propagation of a 1-needle. (c) $B_1(\eta_p(p_1), \eta_q(q_1))$.

As shown by Aichholzer et al. [1], the wavefront W_p changes its propagation speed only if it hits a vertex in $A(p)$. Since the shape of W_p can become very complex after it hits multiple activation points, we make the following simplification for the remainder of this paper: if a wavefront W_p touches a point $q \in A(p)$ we do not change the propagation speed of W_p. Instead, we start a new wavefront at q, which, in turn, starts new wavefronts at points in $A(p)$ if it reaches them earlier than any other wavefront. Hereafter, the start of the propagation of a new wavefront is called an *activation event*, or we say a wavefront is *activated*. The shape of such a new wavefront depends on the position of q on the transportation network. It can be categorized into one of three different shapes: 1-needle, 2-needle, and 3-needle [1] (see Fig. 2). To simplify things, we treat a 2-needle (3-needle) as two (three) 1-needles (see Fig. 3(a)).

When a 1-needle reaches the end of the corresponding network segment, as shown in Fig. 3(b), its shape changes (permanently) [1]. In order to interpret the propagation of a 1-needle, Bae et al. [6] introduced a structure called *needle*. A needle $\eta_p(q, q')$ is a network segment $\overline{qq'}$ with weight $d_C(p, q)$, where $p \in S$ and $q, q' \in A(p)$. Propagating a wavefront from $\eta_p(q, q')$ is equivalent to propagating a 1-needle from q on the network segment $\overline{qq'}$ at time $d_C(p, q)$. If q' is obvious or unimportant we may refer to $\eta_p(q, q')$ as $\eta_p(q)$. Bae et al. also defined the L_1 distance $d_1(\eta_p(q, q'), r)$ between a needle $\eta_p(q, q')$ and a point r as $d_C(p, q)$ plus the length of a quickest path from q to r accelerated by $\overline{qq'}$. Thus, the bisector $B_1(\eta_p(p_1), \eta_q(q_1))$ between two needles $\eta_p(p_1)$ and $\eta_q(q_1)$ is well defined (see Fig. 3(c)).

Shortest Path Map. We use the wavefront model to define shortest path maps [6], and use this concept to explain the formation of *mixed vertices* in Section 3.1, which are important for deriving the structural complexity of the k^{th}-order city Voronoi diagram.

For a site $p \in S$ its shortest path map $\mathcal{SPM}_p$ is a planar subdivision that can be obtained as follows: start by propagating a wavefront from the site p. When a point $q \in A(p)$ is touched for the first time by a wavefront, propagate an additional wavefront from $\eta_p(q)$. Eventually, each point $r \in \mathbb{R}^2$ is touched for the first time by a wavefront propagated from a needle $\eta_p(q)$, where $q \in A(p)$ and $d_1(r, \eta_p(q)) = \min_{q' \in A(p)} d_1(r, \eta_p(q'))$, and q is called the *predecessor* of r. This induces $\mathcal{SPM}_p$. In detail, $\mathcal{SPM}_p$ partitions the plane into at most $|A(p)| = O(c)$ regions $\mathcal{SPM}_p(q)$ such that all points $r \in \mathcal{SPM}_p(q)$ share the same predecessor q and q is on a quickest path from p to r, i.e., $d_C(p, r) = d_C(p, q) + d_C(q, r) = d_1(r, \eta_p(q))$. As proved in [6], the common edge between $\mathcal{SPM}_p(q)$ and $\mathcal{SPM}_p(q')$ where $q, q' \in A(p)$ belongs to the bisector $B_1(\eta_p(q), \eta_p(q'))$. Fig. 4 illustrates an example of the function of shortest path maps where the two Voronoi regions of $V_1(\{p, q\})$ are partitioned by $\mathcal{SPM}_p$ and $\mathcal{SPM}_q$, respectively.

3 Complexity

In this section we derive an upper and a lower bound of the structural complexity of the k^{th}-order city Voronoi diagram $V_k(S)$. In Section 3.1, we first introduce a special degree-2 vertex on a Voronoi edge called *mixed vertex* which is similar to the mixed Voronoi vertices of Cheong et al. [8] for farthest-polygon Voronoi diagrams. Then we derive an upper bound of the structural complexity of $V_k(S)$ in terms of the number of mixed vertices and Voronoi regions. In Section 3.2, we adopt the wavefront concept to introduce a new interpretation for the iterative construction of $V_k(S)$ by Lee [12]. This yields an upper bound for the structural complexity of $V_k(S)$. In Section 3.3 we construct a worst-case example to obtain a lower bound for the structural complexity of $V_k(S)$.

3.1 Mixed Vertices

Definition 1 (Mixed Vertex). *For two sites $p, q \in S$ and a Voronoi edge e which is part of $B_C(p, q)$, a point r on e is* a mixed vertex *if there are $p_1, p_2 \in A(p)$ and $q_1 \in A(q)$ such that $r \in \mathcal{SPM}_p(p_1) \cap \mathcal{SPM}_p(p_2) \cap \mathcal{SPM}_q(q_1)$.*

For instance, Fig. 4 shows a first-order city Voronoi diagram $V_1(\{p, q\})$, where the mixed vertices are marked with a square and denoted by $m_1, \ldots, m_4$. The vertex m_2 is a mixed vertex because it is in $\mathcal{SPM}_p(p_1) \cap \mathcal{SPM}_p(p_2) \cap \mathcal{SPM}_q(q_1)$. Definition 1 yields the following.

Lemma 1. *If a Voronoi edge e contains $m \geq 0$ mixed vertices, its complexity is $O(m + 1)$.*

Lemma 2. *An upper bound for the structural complexity of a k^{th}-order city Voronoi diagram $V_k(S)$ is $O(M + k(n-k))$, where M is the total number of mixed vertices.*

Proof. Lee [12] proved that the number of Voronoi regions in the k^{th}-order Voronoi diagram is $O(k(n-k))$ in any distance metric satisfying the triangle inequality, and so is the number of Voronoi edges. By Lemma 1, if a Voronoi edge e contains m_e mixed Voronoi vertices, its complexity is $O(m_e+1)$. Suppose a city Voronoi diagram $V_k(S)$ contains a set E of Voronoi edges, and each edge $e \in E$ contains m_e mixed Voronoi vertices. Then, the complexity of all edges, i.e., the structural complexity of $V_k(S)$, is $\sum_{e \in E} O(m_e+1) = O(M + |E|)$. Since $|E| = O(k(n-k))$, it follows that $O(M + |E|) = O(M + k(n-k))$. □

For the proof in Section 3.2, we further categorize the mixed vertices. Let m be a mixed vertex on the Voronoi edge between $V_k(H_1, S)$ and $V_k(H_2, S)$, where $H_1 \setminus H_2 = \{p\}$ and $H_2 \setminus H_1 = \{q\}$. We call m an *interior mixed vertex* of $V_k(H_1, S)$ if $m \in \mathcal{SPM}_p(p_1) \cap \mathcal{SPM}_p(p_2) \cap \mathcal{SPM}_q(q_1)$, for some $p_1, p_2 \in A(p)$ and $q_1 \in A(q)$; otherwise, we call m an *exterior mixed vertex* of $V_k(H_1, S)$. For example, in Fig.4 the vertices m_2 and m_4 both are interior mixed vertices of $V_1(\{p\}, \{p,q\})$ and exterior mixed vertices of $V_1(\{q\}, \{p,q\})$.

3.2 Upper Bound

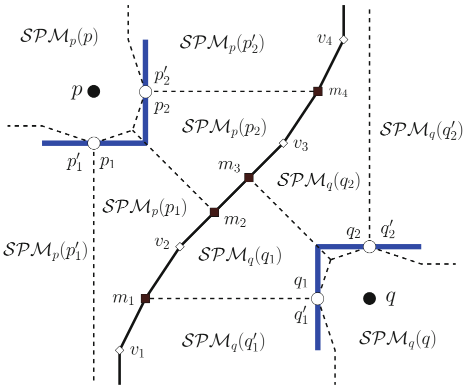


Fig. 4. $B_C(p,q)$ (solid thin edge), where $m_1, \ldots, m_4$ are mixed vertices

Throughout this subsection, we consider a Voronoi region $V_j(H, S)$ of a j^{th}-order Voronoi diagram $V_j(S)$, where $H \subset S$ and $|H| = j$. Let $V_j(H, S)$ have h_H adjacent Voronoi regions $V_j(H_i, S)$ for $1 \leq i \leq h_H$. Note that the subsets H_i and H differ in exactly one element [12]. In the following let $H_i \setminus H = \{q_i\}$, $Q = \{q_1, \ldots, q_{h_H}\}$, and $\ell_H = |Q|$.

Lee [12] proved that in any distance metric satisfying the triangle inequality, $V_j(H, S) \cap V_1(Q) = V_j(H, S) \cap V_{j+1}(S)$, and thus computing $V_1(Q)$ for all the Voronoi regions $V_j(H, S)$ of $V_j(S)$ yields $V_{j+1}(S)$, leading to an iterative construction for $V_k(S)$ for any $k < n$. Fig. 5 illustrates this iteration technique for the Euclidean metric: solid segments form $V_1(H, S)$ and dashed segments form $V_1(Q)$. Since the gray region is part of $V_1(\{p\}, S)$ and also part of $V_1(\{q_1\}, Q)$, all points in the gray region share the same two nearest sites p and q_1, implying that the gray region is part of $V_2(\{p, q_1\}, S)$.

We adopt wavefront propagation to interpret this iterative construction in a new way, which will lead to the main proof of this section. Let us imagine that a

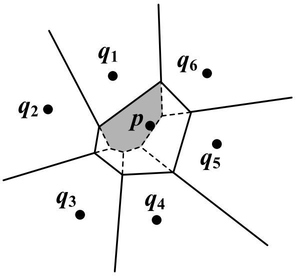


Fig. 5. $V_1(H,S) \cap V_1(Q) = V_1(H,S) \cap V_2(S)$ where $H = \{p\}$, $Q = \{q_1, \ldots, q_6\}$, and $S = H \cup Q$

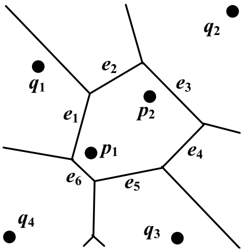


Fig. 6. $V_2(H,S)$ where $H = \{p_1, p_2\}$, $Q = \{q_1, q_2, q_3, q_4\}$, and $S = H \cup Q$

wavefront is propagated from each site $q \in Q$ into the Voronoi region $V_j(H,S)$. If a point $r \in \mathbb{R}^2$ is first touched by the wavefront that propagated from q, r belongs to $V_{j+1}(H \cup \{q\}, S)$.

Note that when $j \geq 2$, $|Q|$ is not necessarily the number of adjacent regions, i.e., $\ell_H \leq h_H$. Fig. 6 illustrates an example for the Euclidean metric: $V_2(H,S)$ has 6 adjacent Voronoi regions but $|Q| = \ell_H$ is only 4. This is because for a site $q \in Q$, $B_2(\{q\},H) \cap V_j(H,S)$ may consist of more than one Voronoi edge, where $B_2(\{q\},H)$ is a Euclidean bisector between $\{q\}$ and H (similar to $B_C(H_1,H_2)$ defined in Section 2). For instance, as shown in Fig. 6, $e_{q_1} = B_2(\{q_1\},H) \cap V_2(H,S)$ consists of two Voronoi edges e_1 and e_2.

Now we transfer our interpretation to the city metric. Let e_q be $B_C(\{q\},H) \cap V_j(H,S)$ for some site $q \in Q$. If e_q contains m_q **exterior** mixed vertices with respect to $V_j(H,S)$, e_q intersects $m_q + 1$ regions in $\mathcal{SPM}_q$. We denote these regions by $\mathcal{SPM}_q(v_z)$ for $1 \leq z \leq m_q + 1$. All v_z must be in $A(q)$. Then, instead of propagating a single wavefront from q into $V_j(H,S)$ (as in the Euclidean metric), we propagate $m_q + 1$ wavefronts, one from each $\eta_q(v_z)$ into $V_j(H,S)$.

As a result, if $V_j(H,S)$ contains m_H exterior mixed vertices, $m_H + \ell_H$ wavefronts will be propagated into $V_j(H,S)$. During the process, when a point $r \in V_j(H,S)$ is *first* touched by a wavefront propagated from $\eta_q(v), q \in Q$ and $v \in A(q)$, we propagate a new wavefront from $\eta_q(r)$, i.e., an activation event occurs, if (i) $r \in \mathcal{P}(V_C) \cup V_C$ or (ii) $v = q$ and $r \in P(q)$. These two conditions amount to $r \in A(q) \setminus \{q\}$, but this classification will help us to derive the number of mixed vertices. This is due to the fact that during the $k-1$ iterations for computing $V_{j+1}(S)$ from $V_j(S)$ for $1 \leq j \leq k-1$, $\mathcal{P}(V_C)$ contributes $O(kc)$ activation events, but $\mathcal{P}(S)$ only contributes $O(n)$.

Lemma 3. *If $V_j(H,S)$ contains m_H exterior mixed vertices, then $V_j(H,S) \cap V_{j+1}(S)$ contains at most $m_H + 2c_H + 2a_H$ mixed vertices, where $c_H = |(\mathcal{P}(V_C) \cup V_C) \cap V_j(H,S)|$ and a_H is the number of activation events associated with points in $\mathcal{P}(S)$.*

Proof. According to the above discussion, we propagate $m_H + \ell_H$ wavefronts into $V_j(H, S)$. All those wavefronts combined generate at most c_H new wavefronts from points in $\mathcal{P}(V_C) \cap V_j(H,S)$, and a_H new wavefronts from points in $\mathcal{P}(S) \cap V_j(H, S)$. Note that $c_H = |(\mathcal{P}(V_C) \cup V_C) \cap V_j(H, S)|$ (condition (i)) but $a_H \leq |\mathcal{P}(S) \cap V_j(H, S)|$ (condition (ii)) Let W be the set of the $m_H + \ell_H + c_H + a_H$ wavefronts. For each point $r \in V_j(H, S)$, if r is first touched by a wavefront $w \in W$ it is associated with w. This will partition $V_j(H, S)$ into $m_H + \ell_H + c_H + a_H$ regions. We view those regions as a special Voronoi diagram $V_1(W)$. Note that $m_H + \ell_H$ of those regions are unbounded.

$V_j(H, S) \cap V_{j+1}(S)$ is a subgraph of $V_1(W)$ since if a point $r \in V_j(H, S)$ is first touched by a wavefront in W propagated from $\eta_q(v)$, r belongs to $V_{j+1}(H \cup \{q\}, S)$. Without loss of generality, we assume every vertex of $V_1(W)$ has degree 3. According to Euler's formula it holds that $N_V = 2(N_R - 1) - N_U$, where N_V, N_R and N_U are the numbers of vertices, regions, and unbounded regions, respectively. Since $V_1(W)$ contains $m_H + \ell_H$ unbounded regions and $m_H + \ell_H + c_H + a_H$ bounded regions, $V_1(W)$ contains $m_H + \ell_H + 2c_H + 2a_H - 2$ vertices. By [12], since $|Q| = \ell_H$, there are $\ell_H - 2$ Voronoi vertices in $V_{j+1}(S) \cap V_j(H, S)$. Therefore, $V_j(H, S) \cap V_{j+1}(S)$ contains at most $(m_H + \ell_H + 2c_H + 2a_H - 2) - (\ell_H - 2) = m_H + 2c_H + 2a_H$ mixed vertices. □

Applying Lemma 3 to each region of $V_j(S)$, yields a recursive formula for the total number of mixed vertices m_{j+1} in $V_{j+1}(S)$: $m_{j+1} = m_j + O(c) + a_j$ (see Lemma 4). In Lemma 5 we show that this formula can be bounded by $O(n + kc)$ for k iterations of this iterative approach. Finally, in Theorem 1 we combine the insights of Lemma 2 and Lemma 5 to give an upper bound for the structural complexity of $V_k(S)$.

Lemma 4. *$V_{j+1}(S)$ contains $m_j + O(c) + 2a_j$ mixed vertices where m_j is the number of mixed vertices of $V_j(S)$ and a_j is the number of activation events associated with points in $\mathcal{P}(S)$ during the computation of $V_{j+1}(S)$ from $V_j(S)$.*

Lemma 5. *The number of mixed vertices of $V_k(S)$ is $O(n + kc)$.*

Theorem 1. *The structural complexity of $V_k(S)$ is $O(k(n - k) + kc)$.*

3.3 Lower Bound

We construct a worst-case example (see Fig. 7) to derive a lower bound for the structural complexity of the k^{th}-order city Voronoi diagram $V_k(S)$. The example consists of a left part and a right part which are placed with a sufficiently large distance between them. We place one vertical network segment in the left part and build a stairlike transportation network in the right part. Then, we place $k + 1$ sites in the right part and the remaining $n - k - 1$ sites in the left part. Since the distance between the left and right part is extremely large, the $n-k-1$ sites in the left part hardly influence the formation of $V_k(S)$ in the right part. Therefore, $V_k(S)$ in the right part forms the farthest-site city Voronoi diagram of the $k + 1$ sites, because sharing the same k nearest sites among $k + 1$ sites is equivalent to sharing the same farthest site among the $k + 1$ sites.

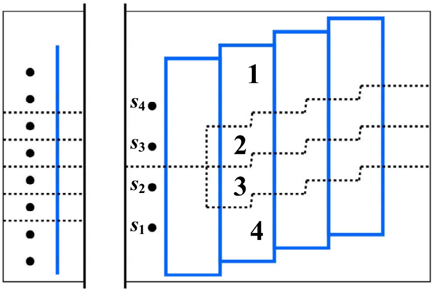


Fig. 7. This worst-case example (here with $k = 3$, $n = 12$, $c = 18$) leads to a lower bound of $\Omega(n+kc)$. The bold solid segments depict the transportation network, and the dashed segments compose $V_k(S)$. The right part is also the farthest-site city Voronoi diagram of $\{s_1, s_2, s_3, s_4\}$, where all points in Region i share the same farthest site s_i.

By construction, as shown in the right part of Fig. 7, all the points in Region i share the same farthest site s_i. Since we can set the speed ν to be large enough, for each point x in Region 2, the shortest path between x and s_1 (s_2) moves along the transportation network counterclockwise, and thus $d_C(s_2, x) > d_C(s_1, x)$. The common Voronoi edge between Regions i and $(i+1)$ contains at least $(\frac{c-2}{4}-1)\cdot 2+1$ (here: 7) segments since the transportation network forms $\frac{c-2}{4}$ rectangles and each rectangle except the first one contains two vertices of the Voronoi edge. Therefore, in the right part, $V_k(S)$ contains at least $(k-1)\frac{c-6}{2} = \Omega(kc)$ segments. Together with the $\Omega(n-k)$ in the left part, we obtain the following lower bound.

Theorem 2. *The structural complexity of $V_k(S)$ is $\Omega(n+kc)$.*

4 Algorithms

In this section we present an iterative algorithm to compute k^{th}-order city Voronoi diagrams in $O(k^2(n+c)\log(n+c))$ time. Its main idea has already been introduced in the complexity considerations in Section 3.2. For the special case of the farthest-site Voronoi diagram, i.e., the $(n-1)^{\text{st}}$-order Voronoi diagram, this algorithm takes $O(n^2(n+c)\log(n+c))$ time. However, for the farthest-site city Voronoi diagram we present a divide-and-conquer algorithm which requires only $O(nc\log^2(n+c)\log n)$ time.

4.1 Iterative Algorithm for k^{th}-Order City Voronoi Diagrams

We describe an algorithm to compute k^{th}-order city Voronoi diagrams $V_k(S)$ based on the ideas in Section 3.2 and Bae et al.'s [6] $O((n+c)\log(n+c))$-time algorithm for the first-order city Voronoi diagram $V_1(S)$. Bae et al.'s approach views each point site in S as a needle with zero-weight and zero-length, and simulates the wavefront propagation from those needles to compute $V_1(S)$. Since their approach can handle general needles, we adopt it to simulate the wavefront propagation of Section 3.2 to compute $V_{j+1}(S)$ from $V_j(S)$.

Algorithm. We give the description of our algorithm for a single Voronoi region $V_j(H, S)$. All four steps have to be repeated for each Voronoi region of $V_j(S)$.

Let $V_j(H,S)$ have h adjacent regions $V_j(H_i,S)$ with $H_i \setminus H = \{q_i\}$ for $1 \leq i \leq h$ and let $Q = \bigcup_{1\leq i\leq h} q_i$. Our algorithm computes $V_j(H,S) \cap V_{j+1}(S)$ as follows:

1. Compute a new set N of sites (needles): For $1 \leq i \leq h$, if the Voronoi edge between $V_j(H_i,S)$ and $V_j(H,S)$ intersect m_i regions $\mathcal{SPM}_{q_i}(v_z)$ in $\mathcal{SPM}_{q_i}$, $1 \leq z \leq m_i$, insert every $\eta_{q_i}(v_z)$ into N.
2. Construct a new transportation network C_H from C: For each point $v \in (\mathcal{P}(V_C) \cup \mathcal{P}(Q) \cup V_C) \cap V_j(H,S)$, if v is located on an edge e of C, insert e into C_H.
3. Perform Bae et al.'s wavefront-based approach to compute $V_1(N)$ under the new transportation network C_H. The approach can intrinsically handle needles as weighted sites.
4. Determine $V_j(H,S)\cap V_{j+1}(S)$ from $V_1(N)$: Consider each edge e in $V_j(H,S)\cap V_1(N)$. Let e be an edge between $V_1(\eta_p(v_p),N)$ and $V_1(\eta_q(v_q),N)$ where $p,q \in S$, $v_p \in A(p)$ and $v_q \in A(q)$. If $p \neq q$, then $e \cap V_j(H,S)$ is part of $V_j(H,S) \cap V_{j+1}(S)$.

Note that Step 2 is used only to reduce the runtime of the algorithm. Lemma 6 shows the correctness and the run time of this algorithm for a single Voronoi region.

Lemma 6. *$V_j(H,S) \cap V_{j+1}(S)$ can be computed in $O((h+m+c_H)\log(n+c))$ time, where h is the number of Voronoi edges, m is the number of mixed vertices, and $c_H = |(\mathcal{P}(V_C) \cup V_C) \cap V_j(H,S)|$.*

Applying Lemma 6 to each region of $V_j(S)$ combined with Theorem 1 leads to Lemma 7. The summation of $O((j(n-j)+jc)\log(n+c))$ in Lemma 7 for $1 \leq j \leq k-1$ gives Theorem 3.

Lemma 7. *$V_{j+1}(S)$ can be computed from $V_j(S)$ in $O((j(n-j)+jc)\log(n+c))$ time.*

Theorem 3. *$V_k(S)$ can be computed in $O(k^2(n+c)\log(n+c))$ time.*

4.2 Divide-and-Conquer Algorithm for Farthest-Site City Voronoi Diagrams

In this section we describe a divide-and-conquer approach to compute the farthest-site city Voronoi diagram $\mathcal{FV}(S)$. Since there are n Voronoi regions in $\mathcal{FV}(S)$ and each of them is associated with a site $p \in S$, we denote such a region by $\mathcal{FV}(p,S)$.

The idea behind this algorithm is as follows: To compute $\mathcal{FV}(S)$, divide S into two equally-sized sets S_1 and $S_2 = S \setminus S_1$, compute $\mathcal{FV}(S_1)$ and $\mathcal{FV}(S_2)$, and then merge the two diagrams into $\mathcal{FV}(S)$. Now, suppose we have already computed $\mathcal{FV}(S_1)$ and $\mathcal{FV}(S_2)$. Then, the edges of a Voronoi region $\mathcal{FV}(p,S)$ in $\mathcal{FV}(S)$ stem from three sources: i) contributed by $\mathcal{FV}(S_1)$, ii) contributed by $\mathcal{FV}(S_2)$, and iii) contributed by two points, one in S_1 and the other in S_2, that have the same distance to two farthest site. In fact, the union of all of the third kind of edges is $B_C(S_1, S_2)$. Each connected component of $B_C(S_1, S_2)$ is called a *merge curve.* A merge curve can be either a closed or open simple curve.

If all the merge curves are computed, merging $\mathcal{FV}(S_1)$ and $\mathcal{FV}(S_2)$ takes time linear in the complexity of $B_C(S_1, S_2)$. To compute the merge curves, we first need to find a point on each merge curve, and then trace out the merge curves from these discovered points.

In order to compute a merge curve, we modify Cheong et al.'s divide-and-conquer algorithm [8] for farthest-polygon Voronoi diagrams in the Euclidean metric to satisfy our requirements. Given a set $\mathcal{P}$ of disjoint polygons, $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$, of total complexity n, the farthest-polygon Voronoi diagram $\mathcal{FV}(\mathcal{P})$ partitions the plane into Voronoi regions such that all points in a Voronoi region share the same farthest polygon in $\mathcal{P}$. Let $|P|$ be the number of vertices of a polygon $P \in \mathcal{P}$ and let $|\mathcal{P}|$ be $\sum_{P \in \mathcal{P}} |P| = n$.

Their algorithm computes the medial-axis $\mathcal{M}(P)$ for each polygon $P \in \mathcal{P}$ and refines $\mathcal{FV}(P, \mathcal{P})$ by $\mathcal{M}(P)$. $\mathcal{M}(P)$ partitions the plane into regions such that all points in a region share the same closest element of P, where an element is a vertex or an edge of P. In other words, for each point $v \in \mathbb{R}^2$, $\mathcal{M}(P)$ provides a shortest path between v and P. Therefore, the medial axes for $\mathcal{FV}(P, \mathcal{P})$, with $P \in \mathcal{P}$, have the same function as the shortest path maps $\mathcal{SPM}_p$, with $p \in S$ in the city metric. By replacing $\mathcal{P}$ and $\mathcal{M}(P)$ with S and $\mathcal{SPM}_p$ respectively, the divide-and-conquer algorithm of Cheong et al. [8] can be modified to compute $\mathcal{FV}(S)$ with respect to the city metric.

Cheong et al. [8] pointed out the bottleneck with respect to running time is to find for each closed merge curve a point that lies on it. In order to overcome the bottleneck, the authors use some specific point location data structures [9,13]. Let $\mathcal{P}$ be divided into two sets $\mathcal{P}_1$ and $\mathcal{P}_2 = \mathcal{P} \setminus \mathcal{P}_1$, where $|\mathcal{P}_1| \approx |\mathcal{P}_2| \approx \frac{n}{2}$. Cheong et al. [8] construct the point location data structures for $\mathcal{FV}(\mathcal{P}_1)$ and $\mathcal{FV}(\mathcal{P}_2)$. For each polygon $P \in \mathcal{P}_1$ and each vertex $v \in \mathcal{M}(P) \cap \mathcal{FV}(P, \mathcal{P}_1)$, they perform a point location query in $\mathcal{FV}(\mathcal{P}_1)$ and $\mathcal{FV}(\mathcal{P}_2)$ (likewise for each polygon $P' \in \mathcal{P}_2$). Each point location query requires $O(\log n)$ primitive operations, and each operation tests for $O(1)$ points and takes $O(\log n)$ time. Hence, one point location query takes $O(\log^2 n)$ time. Since $|\mathcal{FV}(\mathcal{P}_1)| = |\mathcal{FV}(\mathcal{P}_1)| = O(n)$, merging $\mathcal{FV}(\mathcal{P}_1)$ and $\mathcal{FV}(\mathcal{P}_2)$ takes $O(n \log^2 n)$ time.

Since in our case $|\mathcal{FV}(S_1)| = |\mathcal{FV}(S_2)| = O(nc)$, we perform $O(nc)$ point location queries, each of which takes $O(\log^2 nc) = O(\log^2 (n+c)^2) = O(\log^2 (n+c))$ time. Therefore, merging $\mathcal{FV}(S_1)$ and $\mathcal{FV}(S_2)$ takes $O(nc \log^2 (n+c))$ time. We conclude:

Theorem 4. *$\mathcal{FV}(S)$ can be computed in $O(nc \log n \log^2 (n+c))$ time.*

5 Conclusion

We contribute two major results for the k^{th}-order city Voronoi diagram. First, we prove that its structural complexity is $O(k(n-k)+kc)$ and $\Omega(n+kc)$. This is quite different from the $O(k(n-k))$ bound in the Euclidean metric [12]. It is especially noteworthy that when $k = n - 1$, i.e., the farthest-site Voronoi diagram, its structural complexity in the Euclidean metric is $O(n)$, while in the city metric it is $\Theta(nc)$. Secondly, we develop the first algorithms that compute the k^{th}-order city Voronoi diagram and the farthest-site Voronoi diagram. Our algorithms show that traditional techniques can be applied to the city metric.

Acknowledgment. A. Gemsa received financial support by the *Concept for the Future* of KIT within the framework of the German Excellence Initiative. D. T. Lee and C.-H. Liu are supported by the National Science Council, Taiwan under grants No. NSC-98-2221-E-001-007-MY3 and No. NSC-99-2911-I-001-506.

References

1. Aichholzer, O., Aurenhammer, F., Palop, B.: Quickest paths, straight skeletons, and the city Voronoi diagram. Discrete Comput. Geom. 31, 17–35 (2004)
2. Agarwal, P.K., de Berg, M., Matoušek, J., Schwarzkopf, O.: Constructing levels in arrangements and higher order Voronoi diagrams. SIAM J. Comput. 27(3), 654–667 (1998)
3. Aurenhammer, F., Schwarzkopf, O.: A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. Internat. J. Comput. Geom. Appl. 2, 363–381 (1992)
4. Bae, S.W., Chwa, K.-Y.: Shortest Paths and Voronoi Diagrams with Transportation Networks Under General Distances. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1007–1018. Springer, Heidelberg (2005)
5. Boissonnat, J.D., Devillers, O., Teillaud, M.: A semidynamic construction of higher-order Voronoi diagrams and its randomized analysis. Algorithmica 9, 329–356 (1993)
6. Bae, S.W., Kim, J.-H., Chwa, K.-Y.: Optimal construction of the city Voronoi diagram. Internat. J. Comput. Geom. Appl. 19(2), 95–117 (2009)
7. Chazelle, B., Edelsbrunner, H.: An improved algorithm for constructing kth-order Voronoi diagrams. IEEE Transactions on Computers 36(11), 1349–1454 (1987)
8. Cheong, O., Everett, H., Glisse, M., Gudmundsson, J., Hornus, S., Lazard, S., Lee, M., Na, H.-S.: Farthest-polygon Voronoi diagrams. Comput. Geom. Theory Appl. 44, 234–247 (2011)
9. Edelsbrunner, H., Guibas, L.J., Stolfi, J.: Optimal point location in a monotone subdivision. SIAM J. Comput. 15(2), 317–340 (1986)
10. Gemsa, A., Lee, D.T., Liu, C.-H., Wagner, D.: Higher order city Voronoi diagrams. ArXiv e-prints, arXiv:1204.4374 (April 2012)
11. Görke, R., Shin, C.-S., Wolff, A.: Constructing the city Voronoi diagram faster. Internat. J. Comput. Geom. Appl. 18(4), 275–294 (2008)
12. Lee, D.T.: On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Comput. 31(6), 478–487 (1982)
13. Mulmuley, K.: A fast planar partition algorithm, I. J. Symbolic Comput. 10(3-4), 253–280 (1990)
14. Mulmuley, K.: On levels in arrangements and Voronoi diagrams. Discrete Comput. Geom. 6, 307–338 (1991)

On Minimum Sum of Radii and Diameters Clustering

Babak Behsaz and Mohammad R. Salavatipour*

Dept. of Computing Sci., Univ. of Alberta, Edmonton, Alberta T6G 2E8, Canada
{behsaz,mrs}@ualberta.ca

Abstract. Given a metric (V, d) and an integer k, we consider the problem of covering the points of V with at most k clusters so as to minimize the sum of radii or the sum of diameters of these clusters. The former problem is called the Minimum Sum Radii (MSR) problem and the latter is the Minimum Sum Diameters (MSD) problem. The current best polynomial time algorithms for these problems have approximation ratios 3.504 and 7.008, respectively [2]. For the MSR problem, we give an exact algorithm when the metric is the shortest-path metric of an unweighted graph and there cannot be any singleton clusters. For the MSD problem on the plane with Euclidean distances, we present a polynomial time approximation scheme.

Keywords: clustering, minimum sum radii and diameters, Euclidean.

1 Introduction

Clustering is one of the fundamental techniques in information technology. The main goal of this technique is to partition a set of objects into a number of homogeneous subsets, called *clusters*. In any clustering method, we need to define a distance measure between each pair of objects to determine how similar those objects are. In most clustering algorithms, we try to find a clustering that optimizes an objective function based on these distances. The well known k-center problem is the clustering problem with the objective of minimizing the maximum cluster diameter (see [7] for a tight approximation algorithm).

Unfortunately, in some applications, using k-center objective function produces a *dissection effect*. This effect causes objects that should be placed in the same cluster to be assigned to different clusters [6]. To avoid this effect, it is proposed to minimize the sum of cluster radii or diameters. This leads to the Minimum Sum Radii (MSR) and the Minimum Sum Diameters (MSD) problems, respectively. In each of these problems, one is given a set of points V in a metric space d and the goal is to partition V into k clusters so as to minimize the sum of radii of clusters (in MSR) or the sum of diameters of the clusters (in MSD). We can consider these points as the vertices of a graph with a metric

* Research supported by Alberta Innovates Future Technologies, Canada. The second author was additionally supported by NSERC.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 71–82, 2012.

cost function on the edges. More formally, we are given a graph $G = (V, E)$ with edge costs (or distances) $d : E \rightarrow \mathbb{Q}^+$ that satisfies triangle inequality. The goal is to partition V into k sets $V_1, V_2, \ldots, V_k$. In the MSR problem, we want to minimize $\sum_{i=1}^{k} \text{rad}(V_i)$, where $\text{rad}(V_i)$ is equal to $\min_{u \in V_i} \max_{v \in V_i} d(u, v)$. In the MSD problem, we want to minimize $\sum_{i=1}^{k} \text{diam}(V_i)$, where $\text{diam}(V_i)$ is equal to $\max_{u,v \in V_i} d(u, v)$.

Both the MSR and MSD problems are well studied in general and Euclidean metrics. When cost function is symmetric, a simple observation is that for any graph (or cluster) G: $\text{rad}(G) \leq \text{diam}(G) \leq 2\,\text{rad}(G)$. Therefore, an α-approximation algorithm for MSD yields a 2α-approximation algorithm for MSR and vice versa. Doddi *et al.* [3] considered the MSD problem and showed that unless $P = NP$, for any $\epsilon > 0$, there is no $(2 - \epsilon)$-approximation algorithm for the MSD problem even when the graph is unweighed (i.e. the metric is the shortest-path metric of an unweighted graph). Note that this result does not imply NP-hardness of MSR. They also presented a bicriteria algorithm that returns a solution with $O(k)$ clusters whose cost is within $O(\log(n/k))$ factor of the optimum. Charikar and Panigrahy [2] significantly improved this result by giving a $(3.504 + \epsilon)$-approximation algorithm for MSR, and consequently a $(7.008 + \epsilon)$-approximation algorithm for MSD, that runs in time $n^{O(\frac{1}{\epsilon})}$. These are the current best ratios for the MSR and MSD problems on general metrics. Recently, in an interesting result, Gibson *et al.* [4] designed an exact algorithm for the MSR problem which runs in time $n^{O(\log n \log \Delta)}$ where Δ is the ratio of the largest distance over the smallest distance. They translate this result to a quasi-polynomial time approximation scheme for general metrics.

There are also several results for the special cases of these problems. When $k = 2$, the MSD problem is solvable in polynomial time [6]. When k is fixed and the metric is Euclidean, Capoyleas *et al.* [1] present an exact algorithm for MSD. For fixed k and general metrics, the exact algorithm of [3] for MSR implies a 2-approximation for MSD. For Euclidean MSR, there is an exact polynomial time algorithm [5]. This result also extends to L_1 and L_∞ metrics. This also implies a 2-approximation for MSD on Euclidean plane, which is the current best ratio. In contrast, the MSR problem is NP-hard even in metrics induced by weighted planar graphs and in metrics of constant doubling dimension [4].

1.1 Our Results

We know that for graphs with polynomially bounded Δ (the ratio of the heaviest to lightest edge cost), for instance for unweighted graphs, there is an exact algorithm for MSR which runs in time $n^{O(\log^2 n)}$ [4]. This gives us a strong evidence that one may be able to design an exact algorithm for MSR when restricted to these metrics, while the best known polynomial time algorithm is a $(3.504 + \epsilon)$-approximation. We make some progress in this direction and give a polynomial time exact algorithm for metrics of unweighted graphs in the case that no singleton clusters is allowed.

Theorem 1. *There is a polynomial time exact algorithm for the unweighted MSR problem when no singletons (clusters of radius zero) is allowed.*

This result reduces the unweighted MSR problem to the problem of finding the singleton clusters. Moreover, when the running time is critical, we have the following simple algorithm:

Theorem 2. *Finding the best single cluster of each connected component is a* $\frac{3}{2}$*-approximation for the unweighted MSR problem without singletons.*

For Euclidean MSD (*i.e.*, points in $\mathbb{R}^2$ and Euclidean metric), the exact algorithm of Capoyleas *et al.* for fixed k raises a question about the complexity of this problem for variable k (first asked by Doddi *et al.* [3]). Recall that the exact algorithm of [5] gives a 2-approximation for Euclidean MSD and there is ratio 2 hardness for the general case [3]. We give a Polynomial Time Approximation Scheme (PTAS) for the Euclidean MSD:

Theorem 3. *There is an algorithm such that when given a set of* n *points* V *in* $\mathbb{R}^2$*, an integer* k*, and an error bound* $\epsilon > 0$*, finds in time* $n^{O(1/\epsilon)}$ *a* $(1+\epsilon)$*-approximate solution to the MSD problem on the given input.*

Due to lack of space, we differ most of the proofs to the full version. In Section 2, we present the exact algorithm for the unweighted MSR problem when no singleton clusters is allowed. In Section 3, we present a PTAS for the Euclidean MSD problem. Our concluding remarks come in Section 4.

2 MSR Restricted to Unweighted Graphs

In this section, we focus on the MSR problem when the metric is the shortest path metric of an unweighted graph. First, note that if one can optimally solve the MSR problem for some metric in polynomial time for connected graphs, then using a standard dynamic programming approach, one can optimally solve the problem for that metric for all graphs (see the full version for the proof):

Proposition 1. *The MSR problem reduces in polynomial time to the MSR problem for connected graphs.*

As a consequence, we are going to assume that the input graph is connected in the rest of this section. We start with some definitions that we use.

Definition 1. A *ball* of radius r centered at v, denoted by $B(v, r)$, is the set of vertices $\{u \in V : d(v, u) \leq r\}$. We call a ball of radius zero a *zero ball* or a *singleton.* We say two balls *intersect* if they share at least one common vertex. We say two balls are *adjacent* if they do not intersect and there is an edge that connects two vertices of these balls. Among the optimal solutions, a *canonical optimal* solution is a solution with the minimum possible number of balls.

We have the following lemmas about the structure of a canonical optimal solution in an unweighted graph. Note that although in Theorem 1 we assume no singleton clusters is allowed, the following two lemmas hold without this assumption.

Lemma 1. *A canonical optimal solution does not have any intersecting balls.*

Proof. It is enough to show that two intersecting balls can be merged into one ball with radius at most sum of their radii. See the full version for the details. □

Lemma 2. *In a canonical optimal solution, each ball is adjacent to at most two non-zero balls.*

Proof. Let $B(v, r)$ be a ball in a canonical optimal solution and let $B(v_1, r_1)$, $\ldots, B(v_l, r_l)$ be its adjacent balls such that $r_1 \geq r_2 \geq \cdots \geq r_l$ and assume that $l \geq 3$. It is not hard to show that we can find a ball with radius $r+r_1+r_2+1$ that covers $B(v, r)$ and all its adjacent balls (see the full version for the details). As a result, if $r_3 > 0$, we can decrease the number of balls in the canonical optimal solution without increasing the cost, which is a contradiction. □

Suppose (G, k) is an instance of the (unweighted) MSR problem where no zero balls is allowed. Consider a canonical optimal solution $\mathcal{S}^*$ and suppose that we have k^* balls in $\mathcal{S}^*$. Since we can run the algorithm for all values $q \leq k$ and take the best solution of all, for simplicity of exposition we assume $k^* = k$. By Lemmas 1 and 2 and because no zero balls is allowed, the balls in $\mathcal{S}^*$ are disjoint and each has at most two adjacent balls. Therefore, the balls in $\mathcal{S}^*$ form a path or cycle. Assume that these balls form a path, say $B_1^*, B_2^*, \ldots, B_k^*$, where each B_i^* is adjacent to B_{i+1}^*, $1 \leq i < k$ (the case of a cycle reduces to the case of a path as we show shortly). We give an exact algorithm for this case. Let $\mathcal{B}$ be the set of all possible balls. We use the following simple observation made by many authors including Doddi *et al.*:

Observation 1 *[3] The size of $\mathcal{B}$ is at most n^2.*

The general idea of our algorithm is as follows. For every $V' \subseteq V$, let $G[V']$ denote the induced subgraph of G on V'. Let $G_i = G[\cup_{l=1}^{i} B_l^*]$ (i.e. the subgraph of G induced by the vertices in the first i balls of the path). It is easy to see that for all $1 \leq i \leq k$, the solution $B_1^*, B_2^*, ..., B_i^*$ is an optimal solution for covering G_i with i balls. We present a recursive algorithm, called BestCover, that given a graph H and the number of clusters j, returns a feasible solution, and this feasible solution is optimal when $H = G_i$ and $j = i$, for any $1 \leq i \leq k$. For the moment suppose that the algorithm works as described for values of $j < l$ (for some $l \leq k$). When $j = l$, we guess the ball B_l^* (by enumerating over all possible balls in $\mathcal{B}$) and remove this ball to get a new graph H'. Then, run BestCover with parameters H' and $l-1$ and return the union of the guessed ball and the solution of the recursive call. Note that regardless of whether $H = G_l$ or not, assuming that the recursive call returns a feasible solution for H', we have a feasible solution for H. Furthermore, if $H = G_l$ then for the guessed ball (namely B_l^*), $H' = G_{l-1}$ and the solution returned by the recursive call is optimal and has the same cost as the solution $B_1^*, B_2^*, ..., B_{l-1}^*$. Adding the guessed ball B_l^* to the returned solution, we get an optimal solution for G_l.

The difficulty with the above general idea is that even if we use a dynamic programming approach and save the solution of each distinct input of BestCover that we encounter, there may still be exponentially many different inputs corresponding to exponentially many subsets of $V(G)$. We fix this problem with the crucial observation that we are interested to solve only the subproblems corresponding to graphs G_i. Of course, we do not know $\mathcal{S}^*$ and the subsets $\cup_{l=1}^{i} B_l^*$ in advance, but we show that we can find a polynomial size family of subsets of $V(G)$, called candidate family $\mathcal{F}$, that contains subsets $\cup_{l=1}^{i} B_l^*$ for $1 \leq i \leq k$. Then we only solve the problem for graphs induced by subsets in $\mathcal{F}$, which gives the solution for graph G as well, because $V(G) = \cup_{l=1}^{k} B_l^*$ is in $\mathcal{F}$.

Definition 2. A *candidate family* $\mathcal{F}$, is a set of subsets of $V(G)$ which consists of the following sets: a subset $S \subseteq V(G)$ is in $\mathcal{F}$ if $S = V(G)$ or if there exists a ball B such that $G \setminus B$ has at most two connected components and the set of vertices in one of those components is S.

Lemma 3. *A candidate family can be computed in polynomial time, has at most* $2n^2 + 1$ *members and contains subsets* $\cup_{l=1}^{i} B_l^*$ *for all* $1 \leq i \leq k$.

Proof. Recall that $|\mathcal{B}| \leq n^2$. We remove each of these $|\mathcal{B}|$ balls from G. If the number of connected components is at most two, we add the set of vertices in each component to $\mathcal{F}$. We add $V(G)$ to $\mathcal{F}$ as well. This can be done in polynomial time and we must have considered all members of $\mathcal{F}$ after checking all the balls. The number of subsets obtained this way can be at most $2|\mathcal{B}| + 1 \leq 2n^2 + 1$. When in this process, we remove B_i^* for some $1 < i \leq k$, we get at most two connected components. Therefore, we add the set of vertices $\cup_{l=1}^{i-1} B_l^*$ to $\mathcal{F}$ for all $1 < i \leq k$. Also, we added $V(G) = \cup_{l=1}^{k} B_l^*$ to $\mathcal{F}$ as well. Thus, $\mathcal{F}$ contains subsets $\cup_{l=1}^{i} B_l^*$ for all $1 \leq i \leq k$. □

The procedure BestCover and the main algorithm (which makes calls to it) are presented below. The following theorem is a re-statement of Theorem 1 (proof appears in the full version).

Algorithm 1. BestCover(H, l)

```
If $V(H) = \emptyset$, return $\emptyset$.
If $l = 0$ then return "infeasible".
Find $v$ the center of $H$ and $r = \text{rad}(H)$.
If $l = 1$, return $B(v, r)$.
If Table$[V(H), l] \neq \emptyset$, return Table$[V(H), l]$.
for all choices of a ball $B \in \mathcal{B}$ do
    if $V(H) \setminus B \in \mathcal{F}$ then
        Store the union of $B$ and the result of BestCover$(H \setminus B, l - 1)$ in $\mathcal{C}$.
If $\mathcal{C}$ is empty, return $B(v, r)$.
Choose a solution in $\mathcal{C}$ having the minimum cost, store it in Table$[V(H), l]$ and
return it.
```

Theorem 4. *Algorithm 2 is a polynomial time exact algorithm for the unweighted MSR problem when no singletons is allowed.*

Algorithm 2

```
for all choices of a ball $B \in \mathcal{B}$ do
    if $H = G \setminus B$ is connected then
        Compute a candidate family $\mathcal{F}$ for $H$ as in Lemma 3.
        Define an array TABLE. For all $S \in \mathcal{F}$ and for all $1 \leq l \leq k$, set TABLE$[S, l] = \emptyset$.

        for all $1 \leq q \leq k - 1$ do
            Store the union of $B$ and the result of BESTCOVER$(H, q)$ in the set of solutions $\mathcal{C}$.
Add the best single cluster to $\mathcal{C}$. Return the minimum cost solution in $\mathcal{C}$.
```

Corollary 1. *Given the location of singleton balls, there exists an exact algorithm for the unweighted MSR problem.*

3 PTAS for Euclidean MSD

In this section, we present a PTAS for the MSD problem in $\mathbb{R}^2$. Throughout this section, we assume that $\epsilon > 0$ is a given fixed constant. We build upon the framework of Gibson *et al.* [5] and introduce some new ideas to make it work for the MSD problem. Gibson *et al.* present an exact algorithm (called GKKPV from now on) for the MSR problem restricted to Euclidean plane. Since our algorithm follows the same steps as the GKKPV algorithm [5] for MSR, we give a brief overview of that algorithm along with the necessary lemmas for its proof of correctness before presenting our PTAS for MSD.

3.1 The Exact Algorithm of [5] for Euclidean MSR

Consider an instance of Euclidean MSR which consists of a set of points V on the plane along with integer k. Let $\mathcal{D}$ be the set of discs with a center $p \in V$ and radius $|pq|$ for some $q \in V$. Clearly every cluster (disc) of an optimum solution for MSR is from $\mathcal{D}$ (note that $|\mathcal{D}| \leq n^2$). An axis parallel rectangle is called *balanced* if the ratio of its width to length is at least 1/3. The GKKPV algorithm is a dynamic programming algorithm which uses balanced rectangles to solve subproblems (i.e. a set of points to be covered with discs).

A *separator* for a (balanced) rectangle is any line which is perpendicular to its longer side and cuts it in the middle third of its longer side. The algorithm starts with a rectangle containing all the points, denoted by R_0, and recursively "cuts" it into two smaller rectangles by selecting a separator line. A vertical or horizontal line is called *critical* if it either goes through a point $p \in V$ or is tangent to some disk in $\mathcal{D}$. Since all vertical (horizontal) lines between two consecutive vertical (horizontal) critical lines intersect the same set of discs, it is enough to fix an arbitrary separator between two consecutive critical line and only consider these separators. Thus, there are only polynomially many separators to consider. Let $L(R)$ show the separators of a rectangle R from these fixed separators.

The algorithm has a recursive procedure $\mathrm{DC}(R, \kappa, T)$ shown below, which takes as input a rectangle R, an integer $\kappa \geq 0$ and a subset $T \subseteq \mathcal{D}$ and computes an optimum MSR solution using at most κ discs for the set of points in $Q = \{q \in V \cap R| \ q \text{ is not covered by } T\}$. The idea is that T is the set of discs in the optimal solution that intersect R and are chosen in higher levels of recursion. The algorithm calls $\mathrm{DC}(R_0, k, \emptyset)$ to find the best cover for V. The value of the sub-problem for a recursive call is stored in a dynamic programming table $\textsc{Table}[V \cap R, \kappa, T]$. They prove that always $|T| \leq \beta = 424$; this combined with the fact that the number of distinct balanced rectangles R is $O(n^4)$ and $\kappa \in O(n)$, imply that the size of the dynamic programming table (and hence sub-problems to compute) is $O(n^{2\beta+5})$, which is polynomially bounded.

Algorithm 3. Recursive Clustering: $\mathrm{DC}(R, \kappa, T)$

```
1: If $\textsc{Table}[V \cap R, \kappa, T]$ is calculated then return.
2: Let $Q = \{q \in V \cap R| \ q$ is not covered by $T\}$. If $Q = \emptyset$ then $\textsc{Table}[V \cap R, \kappa, T] \leftarrow \emptyset$
   and return.
3: If $\kappa = 0$, let $\textsc{Table}[V \cap R, \kappa, T] \leftarrow \{I\}$ (infeasible) and return.
4: If $|Q| = 1$ let $\textsc{Table}[V \cap R, \kappa, T]$ be the solution with a singleton cluster and return.

5: Let $R'$ be a minimal balanced rectangle containing $V \cap R$.
6: Initialize $\mathcal{D}' \leftarrow \{I\}$.
7: for all choices $\ell \in L(R')$ do
8:    for all choices of a set $\mathcal{D}_0 \subseteq \mathcal{D}$ of size at most 12 that intersect $\ell$ do
9:       for all choices of $\kappa_1, \kappa_2 \geq 0$ with $\kappa_1 + \kappa_2 + |\mathcal{D}_0| \leq \kappa$ do
10:         Let $R_1$ and $R_2$ be the two rectangles obtained from cutting $R'$ by $\ell$. Let $T_1 =$
            $\{D \in T \cup \mathcal{D}_0 : \ D$ intersects $R_1\}$ and $T_2 = \{D \in T \cup \mathcal{D}_0 : \ D$ intersects $R_2\}$.

11:         if $|T_1| \leq \beta$ and $|T_2| \leq \beta$ then
12:            Recursively call $\mathrm{DC}(R_1, \kappa_1, T_1)$ and $\mathrm{DC}(R_2, \kappa_2, T_2)$.
13:            if $\mathrm{cost}(\mathcal{D}_0 \cup \textsc{Table}[V \cap R_1, \kappa_1, T_1] \cup \textsc{Table}[V \cap R_2, \kappa_2, T_2]) < \mathrm{cost}(\mathcal{D}')$
               then
14:               Update $\mathcal{D}' \leftarrow \mathcal{D}_0 \cup \textsc{Table}[V \cap R_1, \kappa_1, T_1] \cup \textsc{Table}[V \cap R_2, \kappa_2, T_2]$.
15: Assign $\textsc{Table}[V \cap R, \kappa, T] \leftarrow \mathcal{D}'$ and return.
```

For the proof of correctness of GKKPV, the authors of [5] prove the following lemmas. The first one is used to show that in Step 8, it is sufficient to only consider subsets of size at most 12.

Lemma 4 (Lemma 2.1 in [5]). *If R is a rectangle containing a set of points $P \subseteq V$ and $\mathcal{O}$ is an optimum solution of P, then there is a separator for R that intersects at most 12 discs in $\mathcal{O}$.*

The next lemma essentially bounds the number of large discs of optimum intersecting a rectangle and is used to show that it is sufficient to only consider the choices of T_1 and T_2 as in Step 11. The proof of this lemma is based on the fact that the centers of discs of an optimum solution cannot contain the centers of other discs; so you cannot pack too many of them close to each other.

Lemma 5 (Lemma 2.2 in [5]). *If $\mathcal{O}$ is an optimum solution for a set of points $P \subseteq \mathbb{R}^2$ and R is a rectangle of length $a > 0$ containing P, then the number of discs in $\mathcal{O}$ of radius at least a that intersect R is at most 40.*

The following lemma puts a lower bound on the distance of separator lines of nested rectangles. Its proof follows easily from the definition of separator lines.

Lemma 6 (Lemma 3.1 in [5]). *If $\mathrm{DC}(R_{t-1}, \cdot, \cdot)$ and $\mathrm{DC}(R_t, \cdot, \cdot)$ are two consecutive nested recursive calls, where ℓ_{t-1} is the separator line chosen for R_{t-1} and R_t is one of the two rectangles obtained from cutting R_t by ℓ_{t-1}, if R_t has length a then the distance between any separator of R_t parallel to ℓ_{t-1} and ℓ_{t-1} is at least $a/3$.*

The main part of the proof of correctness is Lemma 3.2 in [5] which proves inductively the correctness of procedure $\mathrm{DC}(R, \kappa, T)$. The heart of this proof is the induction step in which they also show that $|T_1| \leq \beta$ and $|T_2| \leq \beta$.

3.2 A PTAS for Euclidean MSD

In this section, we present a PTAS for the MSD problem in $\mathbb{R}^2$. Let $\mathcal{P}$ be the set of all convex polygons having all corners from V. For a cluster C, let $\mathrm{CH}(C)$ denote the convex-hull of C. It is easy to see that in an optimum solution if for two clusters C_1, C_2, $\mathrm{CH}(C_1)$ and $\mathrm{CH}(C_2)$ intersect then we can replace the two clusters with one containing all the points in $C_1 \cup C_2$ without increasing the total diameter. Thus, we can assume that in an optimum solution the convex-hulls of the clusters are disjoint; so each cluster can be defined by its convex-hull. This convex-hull belongs to $\mathcal{P}$. For this reason, from now on, when we say a *cluster* of optimum, we consider a convex polygon in $\mathcal{P}$. There are two main difficulties in extending the arguments of GKKPV for MSR to MSD.

First, as we have seen before in the MSR problem, one can bound the number of distinct clusters that can appear in a solution by $O(n^2)$. This allows one to enumerate over all possible clusters (and more generally over all constant size subsets of clusters) that appear in an optimal solution. This fact is critically used in GKKPV algorithm. For the MSD problem, we do not have such a nice (i.e. polynomially bounded) characterization of clusters that can appear in an optimum solution. To overcome this obstacle, we show that for every cluster C, there is a cluster C' whose diameter is at most a factor $(1 + \epsilon)$ of the diameter of C and C' belongs to a set whose size is polynomially bounded. The critical property of C' is that it is simpler to describe: it is determined by $O(1/\epsilon)$ points.

The second difficulty in adapting the GKKPV algorithm is that they show one cannot have too many large clusters close to a rectangle of comparable length in any optimum solution (Lemma 4). To prove this, they use the simple fact that no disc (cluster) in an optimum solution can contain the center of another disc (cluster) as otherwise one can merge the two clusters into one with smaller total radius. In the MSD problem, clusters are not necessarily defined by a disc and there is no notion of center here. We can still argue that in an optimum solution, we cannot have too many large clusters in a bounded region but the packing argument is different from that of [5] (see Lemma 10).

Lemma 7. [8] *A convex polygon in* $\mathbb{R}^2$ *with diameter* D *has an enclosing circle with radius at most* $\sqrt{3}D/3$.

Definition 3. A *σ-restricted polygon* is a convex polygon with at most σ sides such that each side contains (not necessarily at the corners) at least two points of V.

We emphasize that the corners of a σ-restricted polygon is not necessarily a point of V; it is obtained from the intersection of two lines each of which contains at least two points of V.

Lemma 8. *For any given convex polygon* P *and fixed* $\epsilon > 0$, *there is a* $\sigma(\epsilon)$-*restricted polygon* Q *with* $\sigma(\epsilon) = 13 + \frac{4\sqrt{3}\pi}{3}\frac{1}{\epsilon}$, *satisfying the following properties: 1)* Q *contains* P, *2)* $\operatorname{diam}(Q) \leq (1+\epsilon)\operatorname{diam}(P)$, *and 3) for any vertical or horizontal line* ℓ, ℓ *intersects* P *if and only if* ℓ *intersects* Q.

Proof. The sides of Q are extensions of a subset of sides of P that we choose. In other words, given a subset of edges of P, we consider the straight lines containing these edges and the convex polygon defined by these lines is the polygon Q. The number of sides of Q is exactly equal to the size of the chosen subset of edges of P. Clearly, if we choose Q this way, it encloses P and therefore contains P. We show there is a subset of size $\sigma(\epsilon)$ of edges of P, call it S, such that the resulting polygon Q satisfies properties 2) and 3) as well. Let us order the vertices of P in clock-wise order $u_1, u_2, \ldots$ (starting from the left-most corner of P). We traverse the edges (sides) of P in clockwise order and choose some of them along the way, starting with selecting u_1u_2. In order to satisfy property 3) we also make sure that the left-most, top-most, right-most, and bottom-most corners of P are selected to be in Q (along with the two edges defining those corners of P). So first, we choose the (at most) 8 edges defining these (at most) 4 corners of P and let S consists of these edges. The remaining edges added to Q are obtained by traversing on the edges of P (in clock-wise order) starting from u_1u_2 and add them (if not already among those 8 edges) according to the following rules.

Let D be $\operatorname{diam}(P)$. Suppose the last edge (side) chosen was u_iu_{i+1}. We call an edge u_ju_{j+1} ($j \geq i+1$) *dismissible* with respect to u_iu_{i+1} if: 1) its angle, say θ, with u_iu_{i+1} is less than $\pi/2$ and 2) $\operatorname{dist}(u_{i+1}, u_j) \leq \epsilon D/2$ (see Figure 1). Let j be the largest index such that u_ju_{j+1} is dismissible with respect to the last chosen edge and we add this edge to S, and continue until we arrive at u_1u_2. We let Q be the convex polygon defined by the set of straight lines obtained from the edges in S. Note that Q satisfies property 3).

Let R be the perimeter of P. By Lemma 7, P can be enclosed by a circle of perimeter $2\sqrt{3}\pi D/3$ and hence, we have $R \leq 2\sqrt{3}\pi D/3$. It is not hard to see that we choose at most $2\pi/(\pi/2)$ edges, because of the first condition in definition of a dismissible side and we choose at most $R/(\epsilon D/2)$ edges, because of the second condition. Therefore, we choose a total of at most $1 + 4 + \frac{4\sqrt{3}\pi}{3}\frac{1}{\epsilon}$ sides in our traversal. Given that we add at most 8 edges for the left-most, top-most, right-most, and bottom-most vertices of P, we have $|S| \leq 13 + \frac{4\sqrt{3}\pi}{3}\frac{1}{\epsilon}$.

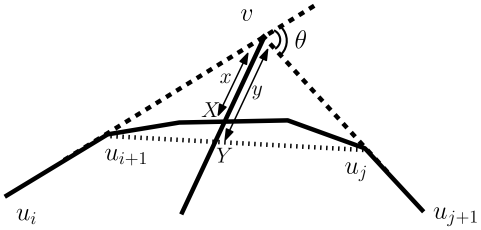


Fig. 1. The solid edges u_iu_{i+1} and u_ju_{j+1} belong to the convex-hull of C. Also, line ℓ is shown with solid line. The extension of edges u_iu_{i+1} and u_ju_{j+1} in polygon Q are shown with dashed lines. The part of l outside P has length x which is less than the length of side $u_{i+1}u_j$.

It only remains to show that $\text{diam}(Q) \leq (1+\epsilon)D$. It is enough to show that for any two corners of Q their distance is at most $(1+\epsilon)D$. Consider two arbitrary corners of Q, say v and w, and the line segment ℓ connecting these points. At most two segments of ℓ may be outside of P and this happens when both v and w are not in corners of P. The section of ℓ that lies inside P has clearly length at most D. We prove that each of the (at most) two segments of ℓ which lies in between P and Q have length at most $\epsilon D/2$. Let us assume that v is the corner obtained from the two lines that contain edges u_iu_{i+1} and u_ju_{j+1} of P and consider the segment of ℓ that has v as an end-point (see Figure 1). Let X be the other end-point of it, which is a cross point of P and ℓ. We also name the cross-point of vw and $u_{i+1}u_j$ point Y. Let x be the length of vX and let y be the length of vY. Because of the convexity of P, we must have $y \geq x$. Consider the triangle $u_{i+1}vu_j$. Since the angle of lines u_iu_{i+1} and u_ju_{j+1}, θ, is less than $\pi/2$, the angle $u_{i+\hat{1}}vu_j \geq \pi/2$, which implies $u_{i+1}u_j$ is the longest side of triangle $u_{i+1}vu_j$. We know that the length of $u_{i+1}u_j$ is at most $\epsilon D/2$. We also know that a segment enclosed in a triangle has length at most equal to the longest edge of the triangle. Thus, we must have $y \leq \epsilon D/2$ which implies $x \leq \epsilon D/2$, as wanted. □

We use OPT to denote an optimal solution. The above lemma implies that for every cluster C of OPT, there is a $\sigma(\epsilon)$-restricted polygon Q containing all the points of C such that $\text{diam}(Q) \leq (1+\epsilon)\,\text{diam}(C)$. Thus, if we replace each cluster C with the set of points in the corresponding $\sigma(\epsilon)$-restricted polygon (breaking the ties for the points that belong to different $\sigma(\epsilon)$-restricted polygons arbitrarily), we find a $(1+\epsilon)$-approximate solution. This enables us to use essentially the same Algorithm 3 (presented for MSR), where we enumerate over $\sigma(\epsilon)$-restricted polygons (instead of all clusters), which is a polynomially bounded set of polygons.

Let R_0 be a rectangle containing the points of V. First, we define a $\Theta(n)$ size set of separators. The notion of balanced rectangle and separator line is the same. A vertical (horizontal) line is called *critical* if it passes through a point in the input. It is not hard to see that all vertical (horizontal) lines between two consecutive critical vertical (horizontal) lines are equivalent in the sense that they intersect the same set of clusters (recall that the clusters are convex polygons). Choose an arbitrary vertical (horizontal) line between each consecutive vertical (horizontal) critical lines. As before, let $L(R)$ show the separators of a rectangle R from these chosen lines. We make the following modification to Algorithm 3. Here, we let $\mathcal{D}$ to be the set of all clusters that correspond to $\sigma(\epsilon)$-restricted polygons, i.e. for each $\sigma(\epsilon)$-restricted polygon P, the set of points in P forms a cluster in $\mathcal{D}$. Obviously, $|\mathcal{D}| \in O(n^{2\sigma(\epsilon)})$ which is polynomial for fixed $\epsilon > 0$. Also, note that for each cluster $C \in \mathcal{P}$, there is a cluster $C' \in \mathcal{D}$ with $\mathrm{diam}(C') \leq (1+\epsilon) \cdot \mathrm{diam}(C)$, by Lemma 8. We fix an arbitrary such cluster C' and call it the representative of C and denote it by $\mathrm{Rep}(C)$. Note that (by Lemma 8):

Corollary 2. *A vertical (or horizontal) line intersects a cluster C if and only if it intersects $\mathrm{Rep}(C)$. In particular for every axis-parallel rectangle R, C intersects R if and only if $\mathrm{Rep}(C)$ intersects R.*

Note that although we have an exponential number of clusters in $\mathcal{P}$, we have a polynomially bounded size set of representatives. For a set of clusters $\mathcal{C}$, we use $\mathrm{Rep}(\mathcal{C})$ to denote the set of clusters in $\mathcal{D}$ that are representative of clusters in $\mathcal{C}$. In particular, $\mathrm{Rep}(\mathrm{OPT})$ are the representative clusters of OPT. We keep the same dynamic programming table, $\textsc{Table}[V \cap R, \kappa, T]$. We will prove that $|T|$ is always a constant. Therefore, given that each cluster of T comes from $\mathcal{D}$, the size of the table is polynomially bounded. Also, instead of a bound of 12 in line 8 for $|\mathcal{D}_0|$, we use a bound of 4 which comes from Lemma 9 which is the equivalent version of Lemma 4 for MSR. In addition, we use $\beta = 83$ for the bounds of $|T_1|$ and $|T_2|$ in line 11. Also, Lemma 10 is the equivalent version of Lemma 5 for MSR . As we stated before, Gibson *et al.*' proof strategy does not work here and we use a new packing argument to prove this lemma. The proof of the following lemmas appear in the full version.

Lemma 9. *Suppose R is a rectangle containing a set of points $P \subseteq V$ and $\mathcal{O}$ is an optimum solution of MSD on P. Let $\mathcal{D}'$ be the set of representatives of clusters of $\mathcal{O}$. Then there is a separator ℓ for R such that it intersects at most 4 clusters of $\mathcal{O}$ (and their representatives in $\mathcal{D}'$).*

Lemma 10. *A rectangle R of length a intersects at most* 3 *clusters of* OPT *with diameter at least a.*

Corollary 3. *A rectangle R of length a intersects at most* 3 *clusters representative of a cluster of diameter at least a in* OPT*.*

Note that Lemma 6 still holds here as it is based on the properties of balanced rectangles and separator lines. We prove the following Lemma which is the equivalent version of Lemma 3.2 in [5].

Lemma 11. *Suppose that* $\mathrm{DC}(R, \kappa, T)$ *is called at some level of recursion. Let* $T' \subseteq \mathrm{OPT}$ *be those clusters of* OPT *that have a point in $V \cap R$ and a point in $V \setminus R$ and suppose* $T = \mathrm{Rep}(T')$*. Also, let* $Q' = \{q \in V \cap R |\ q$ *is not covered by* $T'\}$

and $Q = \{q \in V \cap R |\ q$ *is not covered by* $T\}$*, and* OPT' *be the set of clusters of* OPT *that contain a point in* Q'*. Suppose that* $\kappa \geq |\mathrm{OPT}'|$*. Then after calling* $\mathrm{DC}(R, \kappa, T)$*,* $\textsc{Table}(V \cap R, \kappa, T)$ *contains a* κ*-cover for* Q *whose cost is at most* $(1 + \epsilon) \cdot \mathrm{cost}(\mathrm{OPT}')$*.*

It follows from Lemma 11 that after the call to $\mathrm{DC}(R_0, k, \emptyset)$ the entry $\textsc{Table}[V, k, \emptyset]$ contains an optimum solution for V. This completes the proof of Theorem 3.

4 Concluding Remarks

The exact algorithm for unweighted MSR without singletons (in Section 2) can be used to obtain a $(2+\epsilon)$-approximation for general metric MSR without singletons. The first step is to use a standard scaling [5] so that the aspect ratio of weights are polynomially bounded at a loss of factor $(1 + \epsilon)$. Then one can replace each edge of weight c with a path of c edges and run the algorithm for the unweighted MSR. If a chosen center is not a vertex of the original graph (i.e. is on a path that replaced an edge) then we move that center to the closest vertex of the original graph. Also, the algorithm of Section 3 for Euclidean MSD in $\mathbb{R}^2$ can be extended to a MSD in any fixed dimension Euclidean space. This is similar to the extension of [5] and the details are omitted from this extended abstract.

Recently, we have obtained an exact polynomial time algorithm for the MSD problem with constant k, which settles the open problem of complexity of the MSD problem in this case (first asked by [3]). This will appear in another paper.

There are several interesting open questions concerning MSR and MSD. Perhaps the most interesting one is to obtain a PTAS for general MSR. The existence of a QPTAS for this [4] is a strong evidence that there could be a PTAS. For MSD on Euclidean metrics, the complexity of the problem is still open.

References

1. Capoyleas, V., Rote, G., Woeginger, G.: Geometric clusterings. Journal of Algorithms 12(2), 341–356 (1991)
2. Charikar, M., Panigrahy, R.: Clustering to minimize the sum of cluster diameters. Journal of Computer and System Sciences 68(2), 417–441 (2004)
3. Doddi, S., Marathe, M.V., Ravi, S.S., Taylor, D.S., Widmayer, P.: Approximation algorithms for clustering to minimize the sum of diameters. Nordic J. of Computing 7(3), 185–203 (2000)
4. Gibson, M., Kanade, G., Krohn, E., Pirwani, I.A., Varadarajan, K.: On metric clustering to minimize the sum of radii. Algorithmica (2010)
5. Gibson, M., Kanade, G., Krohn, E., Pirwani, I.A., Varadarajan, K.: On clustering to minimize the sum of radii. SIAM Journal on Computing 41(1), 47–60 (2012)
6. Hansen, P., Jaumard, B.: Minimum sum of diameters clustering. Journal of Classification 4(2), 215–226 (1987)
7. Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Mathematics of Operations Research 10(2), 180–184 (1985)
8. Jung, H.W.: Über die kleinste kugel, die eine räumliche figur einschliesst. J. Reine Angew. Math. 123, 241–257 (1901)

A Simple Framework for the Generalized Nearest Neighbor Problem

Tomas Hruz and Marcel Schöngens

Institute of Theoretical Computer Science, ETH Zurich, Switzerland
{tomas.hruz,schoengens}@inf.ethz.ch

Abstract. The problem of finding a nearest neighbor from a set of points in $\mathbb{R}^d$ to a complex query object has attracted considerable attention due to various applications in computational geometry, bio-informatics, information retrieval, etc. We propose a generic method that solves the problem for various classes of query objects and distance functions in a unified way. Moreover, for linear space requirements the method simplifies the known approach based on ray-shooting in the lower envelope of an arrangement.

1 Introduction

During the last decades the nearest neighbor problem and its variants have attracted considerable attention in computational geometry, information retrieval, pattern recognition, bio-informatics, and many more areas of computer science. In its *classical version*, the problem is stated as follows: Given an n-point set $P \subset \mathbb{R}^d$ together with a metric distance function $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$, preprocess P such that for a query point $\rho \in \mathbb{R}^d$ we can efficiently find a point $\pi \in P$ with $\mathcal{D}(\pi, \rho) \leq \mathcal{D}(\pi', \rho)$ for all $\pi' \in P$. The point π is usually called a *closest point* or a *nearest neighbor* of the query point. There are situations in which we want the query to be an object more complex than a point, as for example a line or simplex in $\mathbb{R}^d$. Consequently, we generalize the notion of a query and define a *query object* Q to be a subset of $\mathbb{R}^d$ that has a constant description complexity. Furthermore, we denote the collection of all allowed query objects by Θ. As an example, consider Θ as the set of all lines in $\mathbb{R}^3$ where a query object $Q \in \Theta$ is the point set of a line. A natural generalization of the problem statement is as follows. Preprocess P such that for a query object $Q \in \Theta$ we can efficiently find a point π in P with $\mathcal{D}(\pi, Q) \leq \mathcal{D}(\pi', Q)$ for all $\pi' \in P$, where $\mathcal{D}(\pi, Q) = \min\{\mathcal{D}(\pi, \rho) \mid \rho \in Q\}$ is the distance between the point π and the query object Q. We refer to this version as the *generalized nearest neighbor problem (GNN problem)*. In this paper we consider the dimension d as a fixed constant.

1.1 Our Contribution

We present a novel, simple framework that solves the generalized nearest neighbor problem uniformly for many natural query objects by a combination of ε-nets

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 83–94, 2012.

and range searching algorithms. Although the results rely on known concepts, we are not aware of any prior work that brings the following contributions:

- Our solution to the GNN problem uses range searching data structures, but does not rely on Megiddo's parametric search [14] that usually causes a logarithmic overhead. Parametric search is a powerful but complex technique to reduce optimization problems to decision problems which we substitute by an application of ε-nets (see Section 2).
- Applying the framework to three sets of query objects improves previously published results: If the query objects are lines in 2-dimensional Euclidean space [15] we improve the query time by a factor of $\mathcal{O}\left(n^{0.195}\right)$. In the case of query planes in 3-dimensional Euclidean space [16] and in the case of query circles [17] we improve two recent results by reducing the space requirements by a logarithmic factor (see Section 3).
- In the case of query lines in 3-space with Manhattan and Euclidean distance, a problem that has not been considered before, we show that, for a parameter $f > 0$, our framework obtains a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ while using linear space (see Section 3).
- The current state-of-the-art idea to approach the GNN problem, which is referred to as generalized Voronoi diagrams, exploits duality properties of the distance function. In the case of linear space requirements our framework simplifies this approach by avoiding any consideration of duality (see Section 4). Furthermore, as opposed to the concept provided by Voronoi diagrams, we give easy to derive bounds on query time and space requirements (see Section 2).

Furthermore, our method is intuitive, natural and could also be applied for the classical nearest neighbor problem. From a practical point of view, the method can use range searching data structures as a black-box.

1.2 Intuitive Idea of Our Approach

When designing a data structure to solve the GNN problem we have a trade-off between space and query time that manifests in two extreme cases: Either we want logarithmic query time and accept larger space requirements, or we aim for linear space but accept larger, still sub-linear, query time. It is common and reasonable to focus on these extreme cases, since space/query time trade-offs in between can usually be derived by a combination of the respective data structures [11]. In this paper we focus on linear space data structures.

Our framework relies on the existence of data structures for the range searching problem that can be formulated as follows [7]: Preprocess a set $P \subset \mathbb{R}^d$ of n points, such that for a given query range $R \subset \mathbb{R}^d$ one can efficiently report all points in $P \cap R$. Now, consider a concrete nearest neighbor problem for a query collection Θ and a distance function $\mathcal{D}$. Let us define the range $\mathcal{B}_r(Q)$ as the set of points within distance r from Q. Observe that this range has the property that for a large enough value of r it contains a nearest neighbor of Q. For example, when the query objects are lines in Euclidean 3-space, for a query Q the range $\mathcal{B}_r(Q)$ is a cylinder with axis Q and radius r (see Section 3). Since known range searching data structures can efficiently report all points in $P \cap \mathcal{B}_r(Q)$, the only problem to identify a nearest neighbor is to find a value r^* such that the nearest neighbor lies in $P \cap \mathcal{B}_{r^*}(Q)$, but not too many other points. This is achieved by identifying a candidate nearest neighbor $\alpha \in P$ such that there are only few points in P closer to the query object Q. The candidate point is found by searching a nearest neighbor in a preprocessed ε-net N of P; the definition and properties of ε-nets are introduced in Section 2. Krauthgamer et al. [9] also use ε-nets to bound search space for proximity search, but their definition describes a completely different notion, which does not provide general worst case guarantees.

Intuitively, the method is correct since either α is a closest point to Q or one of the points in $P \cap \mathcal{B}_{r^*}(Q)$. Both settings are checked by the algorithm. The obtained query time is sub-linear and it depends on the time needed to search the set N and the time needed to report and check all points in $P \cap \mathcal{B}_{r^*}(Q)$. To search N one can either perform a linear scan or recursively apply of the above mentioned procedure. The time for the range searching depends on how much space we accept for the data structure. For many natural query objects there are suitable range searching algorithms with space/query time trade-offs in between the above mentioned two extreme cases. The size of the resulting set $P \cap \mathcal{B}_{r^*}(Q)$ is guaranteed to be at most εn which follows from the properties of the ε-net N (see Section 2). The space requirements of the framework are dominated by the space requirements of the range searching data structure. For details on the algorithm and the analysis we refer the reader to Section 2.

1.3 Related Work

Cole and Yap considered the case of 2-dimensional query lines in Euclidean space [6]. They presented an algorithm which preprocesses n points in $\mathcal{O}\left(n^2\right)$ time and space such that a point closest to a query line can be found in $\mathcal{O}(\log n)$ time. The problem was later reconsidered for space/query time trade-offs: Mitra et al. [15] presented an algorithm with $\mathcal{O}(n \log n)$ preprocessing time, using $\mathcal{O}(n)$ space and $\mathcal{O}\left(n^{0.695}\right)$ query time, and Mukhopadhyay [18] provided an algorithm based on the Partition Theorem [11] with preprocessing time in $\mathcal{O}\left(n^{1+f}\right)$, space requirements in $\mathcal{O}(n \log n)$ and a query time in $\mathcal{O}\left(n^{1/2+f}\right)$ for any $f > 0$.

Another type of query objects, namely planes in Euclidean 3-space, was studied by Mitra et al. [16] who provided an algorithm with $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time, $\mathcal{O}(n \log n)$ space and $\mathcal{O}\left(n^{2/3+f}\right)$ query time for any $f > 0$. The underlying data structure is also based on Matoušek's partition theorem [11].

For the case in which queries are disc boundaries in 2-dimensional Euclidean space, the authors of [17] provided two algorithms that give space/ query time trade-offs: the first has $\mathcal{O}\left(n^3\right)$ preprocessing time and space, and $\mathcal{O}\left(\log^2 n\right)$ query time, and the second has $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time, $\mathcal{O}(n \log n)$ space and $\mathcal{O}\left(n^{2/3+f}\right)$ query time.

There has also been a general idea how to approach the GNN problem. This approach is based on a generalization of Voronoi diagrams (see e.g. [1]) and it can be understood as a kind of guide to solve GNN related problems. Roughly speaking, the idea is based on ray-shooting in the lower envelope of an arrangement of surfaces that are induced by the distance function $\mathcal{D}$ and the point set P. An application of this idea to the above problems works analogously to the analysis in Section 3.1 and yield similar improvements as our framework. However, the idea of applying Voronoi diagrams for the GNN problem is a high level concept that does not directly imply any space or query time guarantees. In contrast, our framework provides these guarantees (Theorem 2 and 3) while being less complex in the application (Section 2 and 4).

2 A Unified Framework for the GNN Problem

We formally introduce the main concepts in the theory of range searching and its connection to our approach. A *range space* $\mathfrak{R}$ is a tuple $(X, \mathit{\Gamma})$, where X is a set and $\mathit{\Gamma}$ is a collection of subsets of X. The elements of X are called points and the elements of $\mathit{\Gamma}$ are called *ranges*. An example for a well-studied range space is $(\mathbb{R}^d, \mathit{\Gamma}_H)$, where $\mathit{\Gamma}_H$ is the set of all half-spaces. The *range searching problem* for a range space $(X, \mathit{\Gamma})$ and a finite point set $P \subseteq X$ can be stated as follows: Preprocess P such that one can efficiently answer the following query. For a range $R \in \mathit{\Gamma}$, report all points in $R \cap P$. This formulation is the so-called reporting version of the problem [2,5]. There is also the counting version, where one is interested in computing $|R \cap P|$. The *restriction of* $\mathit{\Gamma}$ *to a set* $Y \subseteq X$, denoted by $\mathit{\Gamma}|_Y$, is the set $\{Y \cap R \mid R \in \mathit{\Gamma}\}$. An important measure for the complexity of a range space is its *VC-dimension*: A range space $\mathfrak{R}$ has VC-dimension z if there exists a subset $Y \subseteq X$ of maximal cardinality z such that $\mathit{\Gamma}|_Y$ equals the power-set of Y [8]. Though range spaces are formulated on a set theoretic level, in this paper we only need the special case for which $X = \mathbb{R}^d$.

As a first step to use range searching, we need to define an appropriate range space $(\mathbb{R}^d, \mathit{\Gamma})$ for a collection $\mathit{\Theta}$ of query objects and a distance function $\mathcal{D}$. The *Minkowski sum* of two sets A and B is $A + B = \{\alpha + \beta \mid \alpha \in A, \beta \in B\}$. By $\mathcal{B}_r(X)$ we denote the *r-neighborhood* of a set $X \subset \mathbb{R}^d$, which equals the open set $X + \left\{\rho \in \mathbb{R}^d \mid \mathcal{D}(\rho, 0) < r\right\} = \left\{\rho \in \mathbb{R}^d \mid \mathcal{D}(\rho, X) < r\right\}$. The r-neighborhood $\mathcal{B}_r(Q)$ of all $Q \in \mathit{\Theta}$ is a natural set for a range in $\mathit{\Gamma}$, and consequently, we define the desired range space to be $\mathfrak{R}_{\mathit{\Theta}} = (\mathbb{R}^d,\ \{\mathcal{B}_r(Q) \mid Q \in \mathit{\Theta},\ r \geq 0\})$. For example, consider the query collection of all lines in Euclidean 3-space. The 1-neighborhood of a line ℓ is the Minkowski sum of ℓ with the Euclidean unit ball centered at the origin. This forms a cylinder of radius 1 with axis ℓ. Our framework depends on the existence of an efficient range searching algorithm for $\mathfrak{R}_{\mathit{\Theta}}$.

When performing range queries, the resulting sets should not be too large, as otherwise the performance degenerates. We obtain the desired small sets by using ε-nets: Let P be an n-point set of the range space $(X, \mathit{\Gamma})$ and let $\varepsilon \leq 1/2$. A subset N of P is an *ε-net of P for* $(X, \mathit{\Gamma})$ if the following holds. For all

$R \in \Gamma$, if $|R \cap P| \geq \varepsilon |P|$ then $R \cap N \neq \emptyset$. This concept was first introduced to computational geometry by Haussler and Welzl [8], but had also significant impact on other fields of computer science. To simplify notation an ε-net of P is denoted by $\mathcal{N}_\varepsilon(P)$. A notable fact is that for range spaces of finite VC-dimension one can always find ε-nets of P with a size that depends only on ε and not the size of P as the following lemma states.

Lemma 1 (see e.g. [12]). *Let (X, Γ) be a range space with finite VC-dimension $z \geq 1$. For a constant c_z, a parameter $\varepsilon \leq 1/2$ and an n-point subset P of X, there exists an ε-net of P for (X, Γ) of size at most $(c_z/\varepsilon)\log(1/\varepsilon)$.*

2.1 The Framework

We present our framework for the generalized nearest neighbor problem, which we call *GNN-framework* for short. Let $P \subseteq \mathbb{R}^d$ be an n-point set, Θ the collection of query objects, and $\mathcal{D}$ the underlying metric distance function. The query collection together with the distance function form the range space $\mathfrak{R}_\Theta = \big(\mathbb{R}^d, \{\mathcal{B}_r(Q) \mid Q \in \Theta, r \geq 0\}\big)$, which is a fundamental element of the described framework. The framework solves the GNN problem if the range space $\mathfrak{R}_\Theta$ satisfies two properties:

1. There is an algorithm $\mathcal{A}_\mathcal{N}$ that constructs small ε-nets for $\mathfrak{R}_\Theta$
2. There is a reasonable efficient range searching algorithm $\mathcal{A}_\mathfrak{R}$ for $\mathfrak{R}_\Theta$.

These are the only limitations of the framework. Due to Lemma 1 small ε-nets exist for range spaces of finite VC-dimensionality and can be found either by random sampling or deterministically [4]. The existence of an efficient range searching algorithm is not implied by finite VC-dimensionality, but for many natural range spaces efficient algorithms have been found. For example, range spaces that are defined by a constant number of bounded polynomials have been studied by Agarwal et al. [2]. This indicates the potential for solving various concrete instances of the GNN problem as shown in Section 3. The GNN-framework works as follows:

Preprocessing. We preprocess $P_0 = P$ into a data structure $\mathfrak{D}$ which is a $(k+1)$-tuple $((P_0, \mathfrak{S}_0), (P_1, \mathfrak{S}_2), \ldots, (P_k, \emptyset))$ of the following elements. The sets P_i are hierarchically constructed ε-nets for the range space $\mathfrak{R}_\Theta$ that are build by $\mathcal{A}_\mathcal{N}$ in the following way: For a parameter $a \in (0, 1/2)$ we define $\varepsilon_i = n^a / |P_{i-1}|$ and $P_i = \mathcal{N}_{\varepsilon_i}(P_{i-1})$ for $1 \leq i \leq k$. The choice of a depends on the application and influences the query time; We only require that a is chosen such that $|P_i| < |\mathcal{N}_{\varepsilon_i}(P_{i-1})|$. The $\mathfrak{S}_i$ are range searching data structures for $\mathfrak{R}_\Theta$ build by $\mathcal{A}_\mathfrak{R}$ on the sets P_i for $0 \leq i < k$. The parameter k also depends on the application and has only impact on the query time.

Query processing. (see Algorithm 1) For a query object $Q \in \Theta$ we describe how a closest point in P_0 is found. Note that the algorithm works in a recursive fashion and each recursive instance has access to $\mathfrak{D}$ while processing the query. The initial call of the algorithm works on $(P_0, \mathfrak{S}_0)$, the i-th recursive call works on $(P_i, \mathfrak{S}_i)$ and the recursion stops at the k-th recursive call. We describe the initial call of the algorithm since it is representative for the others: First, the algorithm finds a point α closest to Q in the ε-net P_1 by recursively calling itself on P_1. Then, the distance r from α to Q is used to define a range $\mathcal{B}_r(Q) = \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$. The range searching algorithm $\mathcal{A}_{\mathfrak{R}}$ utilizing the data structure $\mathfrak{S}_0$ is asked to retrieve the points $R = P_0 \cap \mathcal{B}_r(Q)$. The resulting set R is searched point-by-point for the nearest neighbor of Q. If the resulting set is empty, α is outputted. As stated above, the last recursive call at recursion depth k is handled specially: The set P_k is searched point-by-point for a nearest neighbor of Q.

Algorithm 1. *Data Structure:* $\mathfrak{D} = ((P_0, \mathfrak{S}_0), (P_1, \mathfrak{S}_1), \ldots, (P_k, \emptyset))$

Initial call: `Query`(Q, 0) *Input:* query object Q *Output:* nearest neighbor of Q;
Comment: `RangeSearch`$(\mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q), i)$ returns $\mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q) \cap P_i$;

```
1  Function: Query($Q$, $i$)
2      If $i < k$ then
3          $\alpha$ = Query($Q, i+1$)
4          $R$ = RangeSearch($\mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q), i$)
5      else
6          $R = P_k$
7      end
8      For each $\pi \in R$ do
9          If $\mathcal{D}(\alpha, Q) \geq \mathcal{D}(\pi, Q)$ then
10             $\alpha = \pi$
11         end
12     end
13
14     return $\alpha$
```

Theorem 1. *Let P be an n-point set, Θ a query collection and $\mathcal{D}$ a distance function. For a query object $Q \in \Theta$, Algorithm 1 correctly outputs a point $\pi \in P$ with $\mathcal{D}(\pi, Q) \leq \mathcal{D}(\pi', Q)$ for all $\pi' \in P$.*

Proof. We prove the theorem by induction on the recursion depth. All line numbers refer to Algorithm 1. As induction basis we focus on the highest level of recursion, where $i = k$. In this case R is set to P_k (line 6) and the for-loop (line 8-12) finds a point α in $P_k = R$ that is closest to Q. This point is returned, so the last recursive call of the algorithm returns a point in P_k that is closest to Q.

As induction hypothesis, we assume that at recursion depth $i+1$ the algorithm returns a point in P_{i+1} that is closest to Q.

For the induction step let $i < k$. The point returned by the recursive call is α (line 3), and by the induction hypothesis it is a point in P_{i+1} closest to Q. The range searching algorithm returns all points in $P_i \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$, consequently all the points in the $\mathcal{D}(\alpha, Q)$-neighborhood of P_i are put in R. If R is non-empty it must contain the desired point and the for-loop (line 8-12) finds it. If R is empty, the point α, which is a point in P_i and P_{i+1}, is closest to Q. Thus the algorithm works correctly.

To bound query time and space requirements, we introduce further notation for the time and space requirements of the subroutines. For the range searching algorithm $\mathcal{A}_{\mathfrak{R}}$ let $\mathcal{T}^*_{\mathfrak{R}}(n)$ be the time needed to preprocess an n-point set P into a range searching data structure of size $\mathcal{S}_{\mathfrak{R}}(n)$, and let $\mathcal{T}_{\mathfrak{R}}(n, m)$ be the time needed to report all m points in the intersection of P with any query range. Furthermore, let $\mathcal{T}_{\mathcal{N}}^{*}(n)$ be the time needed by the ε-net algorithm $\mathcal{A}_{\mathcal{N}}$. The time needed to compute the distance between the query object and a point is at most $\mathcal{T}_{\mathcal{D}}$. We proceed with a bound on the query time.

Theorem 2. *Let P be an n-point set, Θ a query collection and $\mathcal{D}$ a distance function. Furthermore, let $a \in (0, 1/2]$ and $k \in \mathbb{N}$ be parameters of the data structure $\mathfrak{D}$ described above. Then, Algorithm 1 using Algorithm $\mathcal{A}_{\mathfrak{R}}$ finds a nearest neighbor of $Q \in \Theta$ in time $\mathcal{T}(n) \leq k\mathcal{T}_{\mathfrak{R}}(n, n^a) + kn^a\mathcal{T}_{\mathcal{D}} + |P_k|\,\mathcal{T}_{\mathcal{D}}$.*

Proof. We prove the theorem by solving a recurrence equation that we derive from the query algorithm. All line numbers refer to Algorithm 1. At the highest depth of recursion the algorithm works on $(P_k, \emptyset)$ and searches P_k for a closest point to Q (line 6, line 8-12). Thus, we get $\mathcal{T}(|P_k|) = \mathcal{T}_{\mathcal{D}}\,|P_k|$. The running time $\mathcal{T}(|P_i|)$ of all recursive calls at depth $i < k$ working on $(P_i, \mathfrak{S}_i)$ is bounded by the following components. First, by $\mathcal{T}(|P_{i+1}|)$ which is the running time of the recursive call on P_{i+1} (line 3). Then, by $\mathcal{T}_{\mathfrak{R}}(|P_i|, |R|)$ which is the running time of the range searching algorithm $\mathcal{A}_{\mathfrak{R}}$ on P_i, and finally, by the time needed to search the resulting set R for the closest point to Q which is bounded by $\mathcal{T}_{\mathcal{D}}\,|R|$ (line 8-12). From this we get

$$\mathcal{T}(|P_i|) \leq \mathcal{T}(|P_{i+1}|) + \mathcal{T}_{\mathfrak{R}}(|P_i|, |R|) + \mathcal{T}_{\mathcal{D}}\,|R|,$$

for $i < k$. The size of the resulting set $R = P_i \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$ can be bounded by the following arguments. Since α is point in P_{i+1} closest to Q, by the definition of $\mathcal{B}_r(.)$ we deduce that $P_{i+1} \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$ is empty. The set P_{i+1} is an $(n^a/|P_i|)$-net of P_i. Thus, by the definition of ε-nets the size of $R = P_i \cap \mathcal{B}_{\mathcal{D}(\alpha,Q)}(Q)$ is at most $(n^a/|P_i|)\,|P_i| = n^a$. This yields

$$\mathcal{T}(|P_i|) \leq \mathcal{T}(|P_{i+1}|) + \mathcal{T}_{\mathfrak{R}}(|P_i|, n^a) + n^a\mathcal{T}_{\mathcal{D}}.$$

We solve this recurrence equation from $i = 0$ up to k and obtain

$$\mathcal{T}(|P_0|) \leq \sum_{i=0}^{k-1} \left(\mathcal{T}_{\mathfrak{R}}(|P_i|, n^a) + n^a\mathcal{T}_{\mathcal{D}}\right) + |P_k|\,\mathcal{T}_{\mathcal{D}}$$

We have chosen a such that the size of P_i is monotone decreasing in i, thus by $|P_i| \leq |P_0| = n$ we get $\mathcal{T}(n) \leq k\mathcal{T}_{\mathfrak{R}}(n, n^a) + kn^a\mathcal{T}_{\mathcal{D}} + |P_k|\,\mathcal{T}_{\mathcal{D}}$, which proves the theorem.

Theorem 3. *Let P be an n-point set, Θ a query collection and $\mathcal{D}$ a distance function. Furthermore, let $a \in (0, 1/2]$ and $k \in \mathbb{N}$ be parameters of the data structure $\mathfrak{D}$ described above. The time needed to preprocess P into $\mathfrak{D}$ using algorithms $\mathcal{A}_{\mathcal{N}}$ and $\mathcal{A}_{\mathfrak{R}}$ is $\mathcal{T}^{*}(n) \leq k\left(\mathcal{T}_{\mathfrak{R}}^{*}(n) + \mathcal{T}_{\mathcal{N}}^{*}(n)\right)$ and the space requirement is $\mathcal{S}(n) \leq k(n + \mathcal{S}_{\mathfrak{R}}(n))$.*

Proof. The parameter a used for the construction of the ε-nets $P_1, P_2, \ldots, P_k$ is chosen such that the size of P_i is monotone decreasing in i. Both, the time

$\mathcal{T}^*_{\mathcal{N}}(n)$ to create an ε-net on n points as well as the time $\mathcal{T}^*_{\mathfrak{R}}(n)$ increase in n. Thus, the preprocessing time is

$$\mathcal{T}^*(n) \leq \sum_{i=0}^{k} (\mathcal{T}^*_{\mathfrak{R}}(|P_i|) + \mathcal{T}^*_{\mathcal{N}}(|P_i|)) \leq k\,(\mathcal{T}^*_{\mathfrak{R}}(|P_0|) + \mathcal{T}^*_{\mathcal{N}}(|P_0|))\,,$$

which proves the first part of the theorem.

The space requirements $\mathcal{S}_{\mathfrak{R}}(n)$ of the range searching data structure is monotone increasing in n. Hence the space requirement of $\mathfrak{D}$ is

$$\mathcal{S}(n) \leq \sum_{i=0}^{k} (|P_i| + \mathcal{S}_{\mathfrak{R}}(|P_i|)) \leq k(|P_0| + \mathcal{S}_{\mathfrak{R}}(|P_0|)),$$

which proves the second part of the theorem.

We observe that setting the recursion depth k larger than 1 is important when searching the points of the ε-net takes more time than the range searching.

3 Solutions for Concrete Query Collections

In this section we consider several types of query objects and distance functions, some which have been considered before (Section 3.3) and some which we analyze for the first time (Section 3.1 and 3.2). During the explanation we also follow the ideas of Voronoi diagrams to provide a better understanding of the similarities and differences to our framework (Section 3.1). Moreover, in the case of the ℓ_1-distance, we show that the details of a full analysis needed for the a generalization of Voronoi diagrams can lead to large difficulties, however, our framework stays easily applicable (Section 3.2).

3.1 Query Lines in 3-Dimensional Space under ℓ_2-Norm

The problem of searching an n-point set P in $\mathbb{R}^3$ for a nearest neighbor to a given query line has not been considered in literature. The analysis can be generalized to $\mathbb{R}^d$, however, to stay in a geometrically well studied space we focus on $d = 3$. To compare both concepts, we first solve the problem by following the ideas of Voronoi diagrams and afterwards by applying our framework, which obtains the results in a more direct way.

The idea of generalized Voronoi diagrams is based on ray-shooting in the lower envelope of the arrangement induced by the distance function. This arrangement lives in the space of query objects which can be considered as space dual to the point space. Therefore, the first step is to identify the distance function between a point and a line for a suitable representation of a line in $\mathbb{R}^3$. Let the Euclidean norm be denoted by $\|.\|$, the corresponding metric by $\mathcal{D}$ and standard scalar product by $\langle \rho, \rho' \rangle$. Any line Q can be represented by 4 parameters $\chi_Q = (l_1, l_2, l_3, l_4)$; one possible interpretation for these parameters is that $\alpha = (l_1, l_2, 0)$ and $\beta = (l_3, l_4, 1)$ determine the intersection of the line with the plane through the origin and the plane shifted upwards (w.r.t. the last coordinate) by one. The collection of all query lines is $\Theta_\ell = \big\{\{\alpha + t(\beta - \alpha) \mid t \in \mathbb{R}\} \mid (l_1, l_2, l_3, l_4) \in \mathbb{R}^4\big\}$.

The distance between a line $\alpha + t(\beta - \alpha)$ and a point $\pi \in \mathbb{R}^3$ equals the distance between $(\alpha - \pi) + t(\beta - \alpha)$ and the origin, for $t \in \mathbb{R}$. Let $\delta \in \mathbb{R}^3$ be the vector of shortest distance pointing to the line. It satisfies two properties: First, $\langle \delta, (\beta - \alpha)\rangle = 0$ and secondly, $\delta = (\beta - \pi) + s(\beta - \alpha)$ for some $s \in \mathbb{R}$. Inserting the second into the first property yields $s = -\langle \alpha - \pi, \beta - \alpha\rangle / \|\beta - \alpha\|^2$. After some calculation based on properties of the scalar product, the squared length of δ is $\|\delta\|^2 = \|\alpha - \pi\|^2 - \langle \alpha - \pi, \beta - \alpha\rangle^2 / \|\beta - \alpha\|^2$. We translate this rational function to the following polynomial function in π, χ_Q and denote $r = \|\delta\|^2$:

$$\mathcal{F}(\pi, \chi_Q, r) = \|\alpha - \pi\|^2 \|\beta - \alpha\|^2 - \langle \alpha - \pi, \beta - \alpha\rangle^2 - r\,\|\beta - \alpha\|^2 = 0. \qquad (1)$$

We write $\mathcal{F}_\pi(\chi, r)$ for the polynomial $\mathcal{F}(\pi, \chi, r)$ in which the parameter space is χ, r but π is fixed. We call the space in which the polynomial $\mathcal{F}_\pi$ lives *dual space*. Analogously, we call the space where $\mathcal{F}_{\chi,r}(\pi)$ lives *primal space*. The solutions of $\mathcal{F}_\pi(\chi, r) = 0$ for all $\pi \in P$ form an arrangement of algebraic varieties, which have the property that the first algebraic variety hit by a ray starting from $(\chi_Q, 0)$ going into the direction $(\chi_Q, 1)$ represents a nearest neighbor of the line Q.

As described in the introduction it is not known how to perform ray-shooting in an arrangement with linear space. This is indicated by the fact that the complexity of the lower envelope defined by a $(d-1)$-variate function is generally $\Omega(n^{d-1})$ [20]. Thus, we translate the situation to primal space: Every surface $\mathcal{F}_\pi = 0$ is dual to the point π and a ray $\{(\chi, r) \mid r \geq 0\}$ translates to a family of ranges $\{\mathcal{F}_{\chi,r}(\pi) \mid r \geq 0\}$. If the ray hits the first surface $\mathcal{F}_\pi$ at the point (χ, r^*), then the boundary of the range $\mathcal{F}_{\chi,r^*}$ intersects the point π. So, in primal space the ray-shooting translates to (algebraic) range searching that is solved by using a partitioning tree as data structure [2,20]. Additionally the data structure can be equipped with ϵ-nets for every node to constrain the resulting range search operations [13].

Our method, which has the same asymptotic complexity as the method above, does not consider duality at all: We only need to define the range space $\mathfrak{R}_{\Theta_\ell} = \big(\mathbb{R}^3, \{\mathcal{B}_r(Q) \mid Q \in \Theta_\ell, r \geq 0\}\big)$ of all cylinders around all possible query lines using the algebraic variety $\mathcal{F}_{\chi,r} \leq 0$. On P we construct the range searching data structure of [2] for the range space $\mathfrak{R}_{\Theta_\ell}$ which yields a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ and space requirements in $\mathcal{O}(n)$. Secondly, on the point set P we generate a $(n^{-1/2})$-net N of size $\mathcal{O}\left(n^{0.5}\log n\right)$ (Lemma 1). We directly derive the following Observation:

Observation 1. *An n-point set P from 3-dimensional Euclidean space can be preprocessed in a linear space data structure, such that, for a constant parameter $f > 0$, a closest point to given query line can be found in time $\mathcal{O}\left(n^{2/3+f}\right)$.*

3.2 Query Lines in 3-Dimensional Space under ℓ_1-Norm

The GNN problem in 3-dimensional space with Manhattan distance has not been considered in literature. We show that, when following the ideas of generalized Voronoi diagrams, the details get complicated since the distance function is not described by a polynomial any more. In contrast our framework is easily applicable.

The distance between a line Q represented by χ_Q as above and a point $\pi = (p_1, p_2, p_3)$ is

$$\begin{aligned}\mathcal{D}(\pi, Q) = \min\Bigg\{ & |p_1 - l_3 + p_3(l_1 - l_3)| + |p_2 - l_4 + p_3(l_2 - l_3)|, \\ & \left|p_1 - l_3 + \frac{(p_2 + l_4)(l_1 - l_3)}{l_2 - l_4}\right| + \left|p_3 + \frac{p_2 + l_4}{l_2 - l_4}\right|, \\ & \left|p_2 - l_4 + \frac{(p_1 + l_3)(l_2 - l_4)}{(l_1 - l_3)}\right| + \left|p_3 + \frac{p_1 + l_3}{l_1 - l_3}\right| \Bigg\},\end{aligned}$$

because the distance can be computed as a minimum of ℓ_1-distances between the point π and the points $\sigma_1, \sigma_2, \sigma_3$, where σ_i is the intersection of the line Q with the i-th hyperplane that is orthogonal to the coordinate axis and intersects π. The fact that the distance function is a minimum is not harmful since we can take all three functions into the lower envelope with only constant overhead. The real problem comes from the functions themselves, because they are absolute values of rational functions. Known techniques for the decomposition of the lower envelope induced by such functions cannot be directly applied. Further investigation is needed to see that the intersection of the functions is linear so that a projection yields a suitable Voronoi diagram.

On the other hand, the presented framework is easier to apply: It requires only to identify the range space that is naturally given by constant number of simplices. To obtained a range $\mathcal{B}_r(Q)$, it is necessary to project the points of the r-ball of the l_1-metric to the hyperplane that is orthogonal to the query line. The convex hull of the projected points implicitly describes the range. With a standard simplex range searching data structure [3] we obtain the following result.

Observation 2. *An n-point set P from 3-dimensional space with Manhattan distance can be preprocessed into a linear space data structure, such that, for a constant parameter $f > 0$, a closest point to given query line can be found in time $\mathcal{O}\left(n^{2/3+f}\right)$.*

3.3 Previously Considered Query Collections

In the same way we can easily derive data structures and query time bounds for several kind of query objects. Here, we focus on the Euclidean distance.

The case of 2-dimensional query lines has been considered by Mitra et al. and Mukhopadhyay [15,18] who used ham-sandwich cuts or Matoušek's simplicial partion theorem to solve the problem. For linear space with $\mathcal{O}(n \log n)$ preprocessing time, the query time is $\mathcal{O}\left(n^{0.695}\right)$ and for $\mathcal{O}(n \log n)$ space with $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time, the query time is $\mathcal{O}\left(n^{1/2+f}\right)$, for $f > 0$. Furthermore, the case of query 3-dimensional hyperplanes has been considered by Mitra et al. [16] who obtained a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ using $\mathcal{O}\left(n^{1+f}\right)$ preprocessing time and $\mathcal{O}(n \log n)$ space.

An application of our framework for general query hyperplanes in Euclidean space of dimension d achieves a faster query time while using less space. The required range space contains all possible cuts of two parallel half-spaces. An appropriate standard simplex range searching data structures as e.g.[11] can be used. An epsilon-net can be obtained by random sampling in $\mathcal{O}(n)$ time which, however, leads to a randomized algorithm. At the cost of larger preprocessing time, one could also use deterministic ε-net algorithms [10].

Observation 3. *An n-point set P from d-dimensional Euclidean space can be preprocessed in $\mathcal{O}(n \log n)$ time into a linear space data structure, such that, for a parameter $f > 0$, a closest point to a given query hyperplane can be found in time $\mathcal{O}\left(n^{1-1/d+f}\right)$ w.h.p.*

In [17] the authors consider circles as query objects and obtain a query time of $\mathcal{O}\left(n^{2/3+f}\right)$ and $\mathcal{O}(n \log n)$ space. Using a standard lifting transform, the simplex range searching algorithm from above and sampling for the ε-net, we improve the space requirements by a logarithmic factor.

Observation 4. *An n-point set P from 2-dimensional Euclidean space can be preprocessed in $\mathcal{O}(n \log n)$ time into a linear space data structure, such that, for a parameter $f > 0$, a closest point to given query circle can be found in time $\mathcal{O}\left(n^{2/3+f}\right)$ w.h.p.*

These bounds could also be achieved by following the ideas of generalized Voronoi diagrams, which lead to analogous steps as in Section 3.1, but there is no publication considering such an analysis.

4 Comparison with Generalized Voronoi Diagrams

This discussion has to be omitted due to space constraints, but can be found in the technical report [19].

Acknowledgments. The authors would like to thank Peter Widmayer for stimulating discussions and continuous support of this project. We are also grateful to Jiri Matoušek for several discussions on the duality, arrangements and tricky details.

References

1. Agarwal, P., Sharir, M.: Arrangements and their Applications. Handbook of Computational Geometry, pp. 49–119 (1998)
2. Agarwal, P.K., Matoušek, J.: On Range Searching with Semialgebraic Sets. Discrete and Computational Geometry 11(1), 393–418 (1994)

3. Chan, T.M.: Optimal Partition Trees. In: SCG 2010: Proceedings of the 2010 Annual Symposium on Computational Geometry, pp. 1–10. ACM (2010)
4. Chazelle, B.: The Discrepancy Method. Cambridge University Press (2000)
5. Chazelle, B., Welzl, E.: Quasi-optimal range searching in spaces of finite VC-dimension. Discrete and Computational Geometry 4(1), 467–489 (1989)
6. Cole, R., Yap, C.-K.: Geometric Retrieval Problems. In: FOCS 1983: Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, pp. 112–121 (1983)
7. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry - Algorithms and Applications, 2nd edn. Springer (2000)
8. Haussler, D., Welzl, E.: Epsilon-nets and Simplex Range Queries. In: SCG 1986: Proceedings of the 2nd Annual Symposium on Computational Geometry, p. 71. ACM (1986)
9. Krauthgamer, R., Lee, J.: Navigating Nets: Simple Algorithms for Proximity Search. In: SODA 2004: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 798–807. ACM (2004)
10. Matoušek, J.: Construction of epsilon-Nets. Discrete & Computational Geometry 5, 427–448 (1990)
11. Matoušek, J.: Efficient Partition Trees. Discrete and Computational Geometry 8(1), 315–334 (1992)
12. Matoušek, J.: Geometric Discrepancy. Springer (1999)
13. Matoušek, J., Schwarzkopf, O.: On Ray shooting in Convex Polytopes. Discrete and Computational Geometry 10(1), 215–232 (1993)
14. Megiddo, N.: Applying Parallel Computation Algorithms in the Design of Serial Algorithms. Journal of the ACM 30(4), 852–865 (1983)
15. Mitra, P., Chaudhuri, B.B.: Efficiently computing the closest point to a query line. Pattern Recognition Letters 19(11), 1027–1035 (1998)
16. Mitra, P., Mukhopadhyay, A.: Computing a Closest Point to a Query Hyperplane in Three and Higher Dimensions. In: Kumar, V., Gavrilova, M.L., Tan, C.J.K., L'Ecuyer, P. (eds.) ICCSA 2003. LNCS, vol. 2669, pp. 787–796. Springer, Heidelberg (2003)
17. Mitra, P., Mukhopadhyay, A., Rao, S.V.: Computing the Closest Point to a Circle. In: CCCG 2003: Proceedings of the 15th Canadian Conference on Computational Geometry, pp. 132–135 (2003)
18. Mukhopadhyay, A.: Using simplicial partitions to determine a closest point to a query line. Pattern Recognition Letters 24(12), 1915–1920 (2003)
19. Schöngens, M., Hruz, T.: A Simple Framework for the Generalized Nearest Neighbor Problem. Technical Report 758, Theoretical Computer Science, ETH Zurich (2012)
20. Sharir, M., Shaul, H.: Ray Shooting Amid Balls, Farthest Point from a Line, and Range Emptiness Searching. In: SODA 2005: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 525–534 (2005)

Deterministic Parameterized Connected Vertex Cover[⋆]

Marek Cygan

IDSIA, University of Lugano, Switzerland
marek@idsia.ch

Abstract. In the Connected Vertex Cover problem we are given an undirected graph G together with an integer k and we are to find a subset of vertices X of size at most k, such that X contains at least one end-point of each edge and such that X induces a connected subgraph. For this problem we present a deterministic algorithm running in $O(2^k \mathrm{poly}(n))$ time and polynomial space, improving over the previous-best $O(2.4882^k \mathrm{poly}(n))$ time deterministic algorithm and $O(2^k \mathrm{poly}(n))$ time randomized algorithm. Furthermore, when usage of exponential space is allowed, we present an $O(2^k k(n+m))$ time algorithm that solves a more general variant with real weights.

Finally, we show that in $O(2^k \mathrm{poly}(n))$ time and space one can count the number of connected vertex covers of size at most k, and this time upper bound can not be improved to $O((2-\varepsilon)^k \mathrm{poly}(n))$ for any $\varepsilon > 0$ under the Strong Exponential Time Hypothesis, as shown by Cygan et al. [CCC'12].

1 Introduction

In the classical vertex cover problem we are asked whether there exists a set of at most k vertices, containing at least one end-point of each edge. As a basic problem in graph theory Vertex Cover is extensively studied, together with its natural variants. One of the generalizations of Vertex Cover is the Connected Vertex Cover problem, where a vertex cover is called a *connected vertex cover* if it induces a connected subgraph.

Connected Vertex Cover
Input: An undirected graph $G = (V, E)$ and an integer k.
Parameter: k
Question: Does there exist a connected vertex cover of G of cardinality at most k?

As Connected Vertex Cover is NP-complete we can not hope for polynomial time solutions, however it is possible to efficiently solve the problem for small values of k. Obviously, for any fixed k, we can solve the problem in polynomial time, by trying all n^k possible subsets of vertices. In the parameterized

[⋆] The author is partially supported by ERC Starting Grant NEWNET 279352, NCN grant N206567140 and Foundation for Polish Science.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 95–106, 2012.

complexity setting we are interested in finding algorithms of $f(k)\text{poly}(n)$ running time, for some computable function f.

A few fixed-parameter algorithms were designed for the Connected Vertex Cover problem during the last few years. The fastest deterministic algorithm is due to Binkele-Raible [1] running in $O^*(2.4882^k)$ time, while the fastest (randomized) algorithm is due to Cygan et al. [5] running in $O^*(2^k)$ time, where by O^* we denote the standard big O notation, with polynomial factors omitted. In Table 1 we summarize the history of parameterized algorithms for Connected Vertex Cover.

Table 1. Summary of parameterized algorithms for Connected Vertex Cover

$O^*(6^k)$	Guo et al. [12]
$O^*(3.2361^k)$	Mölle et al. [13]
$O^*(2.9316^k)$	Fernau et al. [10]
$O^*(2.7606^k)$	Mölle et al. [14]
$O^*(2.4882^k)$	Binkele-Raible [1]
$O^*(2^k)$(randomized)	Cygan et al. [5]
$O^*(2^k)$	this paper

Our Results. The main result of this paper is a deterministic algorithm solving Connected Vertex Cover in $O^*(2^k)$ time. Moreover, when we allow exponential space, in the same running time we can solve weighted and counting versions of the Connected Vertex Cover problem, which was not possible with the previously fastest randomized algorithm of [5].

#Connected Vertex Cover (#CVC)
Input: An undirected graph $G = (V, E)$, an integer k.
Parameter: k
Goal: Find the number of connected vertex covers of cardinality at most k.

Weighted Connected Vertex Cover (WCVC)
Input: An undirected graph $G = (V, E)$, a weight function $\omega : V \to \mathbb{R}_+$ and an integer k.
Parameter: k
Goal: Find a minimum weight connected vertex cover of cardinality at most k.

Theorem 1. Weighted Connected Vertex Cover *can be solved in* $O(2^k k(|V| + |E|))$ *time and* $O(2^k k)$ *space.*

Theorem 2. #Connected Vertex Cover *can be solved in* $O(2^k\text{poly}(|V|))$ *time and space.*

Recently Cygan et al [4] have shown that unless the *Strong Exponential Time Hypothesis* (SETH) fails, it is not possible to count the number of connected vertex covers of size at most k in $O^*((2-\varepsilon)^k)$ time, for any constant $\varepsilon > 0$. Consequently our counting algorithm is tight under SETH, which is an example of few parameterized problems with nontrivial solutions for which there exists an evidence of optimality.

When restricted to polynomial space, we prove that the weighted variant can still be solved in $O^*(2^k)$ running time, assuming weights are polynomially bounded integers.

Theorem 3. Weighted Connected Vertex Cover *with polynomially bounded integer weights can be solved in* $O(2^k \mathrm{poly}(n))$ *time and polynomial space.*

Related Work. Vertex Cover is one of the longest studied problem in the parameterized complexity. The currently fastest known parameterized algorithm for the Vertex Cover problem is due to Chen et al., running in $O(1.2738^k + kn)$ time [3]. Recently, new parameterizations of Vertex Cover are considered, when the parameter is $k - |M|$ [16], where M is a maximum cardinality matching, or k − LP, where LP is the optimum value of a natural linear programming relaxation [8,18].

A notion very close to fixed-parameter tractability, or even a subfield of it, is kernelization. We call a polynomial time preprocessing routine a kernel, if given an instance I with parameter k the algorithm produces a single instance I' with parameter k', such that I' is a YES-instance iff I is a YES-instance, and moreover $|I'| + k' \leq g(k)$. It is well known that a problem admits a kernel if and only if it is fixed-parameter tractable, however we are mostly interested in kernelization algorithms with the function g being a polynomial. Unfortunately, for Connected Vertex Cover no polynomial kernel exists as shown by Dom et al. [9], unless NP $\subseteq$ coNP/poly.

Organization. In Section 2 we prove Theorem 1. For the sake of presentation we describe small differences needed to solve the counting variant, that is to prove Theorem 2, in separate Section 3. Next, in Section 4 we prove Theorem 3 and finally, we finish the article with conclusions and open problems in Section 5.

Notation. We use standard graph notation. For a graph G, by $V(G)$ and $E(G)$ we denote its vertex and edge sets, respectively. When it is clear which graph we are describing we use n as the number of its vertices and m as the number of its edges. For $v \in V(G)$, its neighborhood $N(v)$ is defined as $N(v) = \{u : uv \in E(G)\}$, and $N[v] = N(v) \cup \{v\}$ is the closed neighborhood of v. We extend this notation to subsets of vertices: $N[X] = \bigcup_{v \in X} N[v]$ and $N(X) = N[X] \setminus X$. For a set $X \subseteq V(G)$ by $G[X]$ we denote the subgraph of G induced by X. For a set X of vertices or edges of G, by $G \setminus X$ we denote the graph with the vertices or edges of X removed; in case of vertex removal, we remove also all the incident edges. For two subsets of vertices $X, Y \subseteq V$ by $E(X,Y)$ we denote the set of edges with one endpoint in X and the other in Y. In particular by $E(X,X)$ we denote the set of edges with both endpoints in X.

For weighted problems we assume weights are representable reals.

2 Algorithm

In this section we prove Theorem 1. As the starting point we use the iterative compression technique in Section 2.1. As a consequence we are left with a problem, where additionally each instance is equipped with a connected vertex cover Z of size at most $k+2$. In Section 2.2 we show how to take advantage of the set Z by showing a natural algorithm, solving a bipartite Steiner tree problem as a subroutine (described in Section 2.4). The key part of the proof of Theorem 1 is the time complexity analysis of the presented algorithm, which relies on a combinatorial lemma proved in Section 2.3.

2.1 Iterative Compression

We start with a standard technique in the design of parameterized algorithms, that is, iterative compression, introduced by Reed et al. [17]. Iterative compression was also the first step of the Monte Carlo algorithm for Connected Vertex Cover [5].

We define a *compression problem*, where the input additionally contains a connected vertex cover $Z \subseteq V$. The name *compression* might be misleading in our case, since in the problem definition below we are not explicitly interested in compressing the solution, but we want to find a minimum weight connected vertex cover using the size of Z as our structural parameter. In particular not only we use the fact that Z is a vertex cover (which ensures that $V \setminus Z$ is an independent set), but also we use the fact that $G[Z]$ is connected, which is crucial for the time complexity analysis of our algorithm.

Compression Weighted Connected Vertex Cover (Comp-WCVC)
Input: An undirected graph $G = (V, E)$, a weight function $\omega : V \to \mathbb{R}_+$, an integer k and a connected vertex cover $Z \subseteq V$ of G.
Parameter: $|Z|$
Goal: Find a minimum weight connected vertex cover of cardinality at most k.

In Section 2.2 we prove the following lemma providing a parameterized algorithm for the above compression problem.

Lemma 4. Comp-WCVC *can be solved in* $O(2^{|Z|}k(|V| + |E|))$ *time and* $O(2^{|Z|}k)$ *space. Moreover, when the weight function is uniform, we can solve the problem in* $O(2^{|Z|}(|V| + |E|))$ *time and* $O(2^{|Z|})$ *space.*

Having the above lemma we show how to efficiently find a connected vertex cover of size at most k (if it exists).

Lemma 5. *Given an undirected graph* $G = (V, E)$ *and an integer* k *one can find a connected vertex cover of size at most* k*, or verify that it does not exist, in* $O(2^k k(|V| + |E|))$ *time and* $O(2^k)$ *space.*

Proof. First, let us assume that G does not contain isolated vertices, since we can remove them. Moreover we can assume that G is connected, since if G contains at least two connected components (and no isolated vertices) then it can not admit a connected vertex cover of any size. Therefore, let $V = \{v_1, ..., v_n\}$ be an ordering of vertices, such that for each $1 \leq i \leq n$, the graph $G[V_i]$ is connected, where $V_i = \{v_i, \ldots, v_n\}$. For $1 \leq i \leq n$ let G_i be the graph G, with vertices of V_i identified to a single vertex. Alternatively, we can say that G_i comes from a contraction of the set of edges of a spanning tree of $G[V_i]$ (where we remove multiple edges and self-loops). Since Connected Vertex Cover is closed under edge contractions, we infer that if there is no connected vertex cover of size at most k in G_i, then clearly there is no connected vertex cover of size at most k in G.

We are going to construct a sequence of sets $X_i \subseteq V(G_i)$ of size at most k, such that X_i is a connected vertex cover of G_i. First, observe that the set $X_1 = \emptyset$ is a connected vertex cover of G_1 of size at most k. Next, let us consider each value of $i = 2, \ldots, n$ one by one. Observe that there is an edge e in $E(G_i)$, such that the graph G_{i-1} is exactly the graph G_i with the edge e contracted. In particular as e we may take any edge between v_{i-1} and V_i. Let x be the vertex in G_{i-1} which corresponds to the set V_{i-1} and let y be the vertex in G_i corresponding to the set V_i. We claim that $Z = (X_{i-1} \setminus \{x\}) \cup \{v_{i-1}, y\}$ is a connected vertex cover of G_i of size at most $k + 2$. Since $|X_{i-1}| \leq k$ the bound on the size of Z holds. Moreover, since X_{i-1} is a vertex cover of G_{i-1}, the set Z is a vertex cover of G_i. Finally, $G_i[Z]$ is connected, because either x is contained in X_{i-1}, or a neighbour of x belongs to X_{i-1}, or x is an isolated vertex which means that $i = 2$ and then $Z = V(G_2)$ induces a connected subgraph.

If, for a fixed i, we use Lemma 4 for the Comp-WCVC instance (G_i, ω, k, Z), with ω being a uniform unit weight function, then in $O(2^{|Z|}(n+m)) = O(2^k(n+m))$ time and $O(2^{|Z|}) = O(2^k)$ space we can find a set X_i, which is a connected vertex cover of G_i of cardinality at most k, or verify that no connected vertex cover of cardinality at most k in the graph G exists. Since $G_n = G$, the set X_n is a connected vertex cover of G of size at most k, which we can find in $O(2^k n(n + m))$ time, because we use Lemma 4 exactly $n - 1$ times. In order to reduce the polynomial factor from $n(n+m)$ to $k(n+m)$ observe, that if we order the set V, such that the set $\{v_{i-\ell+1}, \ldots, v_n\}$ forms a connected vertex cover of the graph G, then as the set $X_{i-\ell+1}$ we can set a singleton set containing the vertex corresponding to $V_{i-\ell+1}$ and reduce the number of rounds in the inductive process from n to ℓ. However, a simple $O(n+m)$ time 2-approximation of the Connected Vertex Cover problem is known [11], which just takes as the solution the set of internal nodes of a depth first search tree of the given graph. Therefore, assuming a vertex cover of size at most k exists, we can find a connected vertex cover of size at most $2k$ in $O(n + m)$ time and consequently reduce the number of rounds of the inductive process to at most $2k$, which leads to $O(2^k k(n + m))$ time complexity. □

Assuming Lemma 4 by Lemma 5 we can find a connected vertex cover Z of size at most k, if it exists. Afterwards we use the set Z as part of the input for Lemma 4, which proves Theorem 1.

2.2 Compression Algorithm

In this section we present a proof of Lemma 4. The advantage we have while solving Comp-WCVC instead of WCVC is the additional set Z, which forms a connected vertex cover of G and the size of Z is our new parameter. We show how to use the set Z as an insight into the structure of the graph and solve compression problem efficiently. The algorithm itself is straightforward, but the crucial part of its time complexity analysis lies in the following combinatorial bound, which we prove in Section 2.3.

Lemma 6. *For any connected graph $G = (V, E)$ we have*

$$\sum_{\substack{V_1 \subseteq V \\ E(G[V \setminus V_1]) = \emptyset}} 2^{|\mathrm{cc}(G[V_1])|} \leq 3 \cdot 2^{|V|-1}, \tag{1}$$

where by $\mathrm{cc}(H)$ we denote the set of connected components of a graph H.

Observe that, in the above lemma, we sum over all sets V_1, that form a vertex cover of G. The second tool we use in the proof of Lemma 4 is the following lemma solving the node-weighted Steiner tree problem in bipartite graphs, where both the terminals and non-terminals form independent sets. The proof of it can be found in Section 2.4.

Lemma 7. *Let $G = (V, E)$ be a bipartite graph and $T \subseteq V$ be a set of terminals, such that T and $V \setminus T$ are independent sets. For a given weight function $\omega : V \setminus T \to \mathbb{R}_+$ and an integer k in $O(2^{|T|}k(|V| + |E|))$ time and $O(2^{|T|}k)$ space we can find a minimum weight subset $X \subseteq V \setminus T$ of cardinality at most k, such that $G[T \cup X]$ is connected, or verify that such a set does not exist. Moreover for a uniform weight function ω we improve the running time to $O(2^{|T|}(|V| + |E|))$ and space usage to $O(2^{|T|})$.*

Having Lemmas 6 and 7 we can prove Lemma 4.

Proof (of Lemma 4). Similarly as in the proof of Lemma 5 we may assume that the graph G is connected. We start with guessing, by trying all $2^{|Z|}$ possibilities, a subset Z_1 of Z that is part of a minimum connected vertex cover and denote $Z_0 = Z \setminus Z_1$.

First, let us consider a special case, that is $Z_1 = \emptyset$. Then we need to take the whole set $V \setminus Z$ to cover the edges $E(Z_1, V \setminus Z)$, since each vertex of $V \setminus Z$ has at least one neighbour in Z (otherwise the vertex would be isolated). It is easy to verify whether $V \setminus Z$ is a connected vertex cover of size at most k.

Therefore, we assume that $Z_1 \neq \emptyset$ and moreover $E(Z_0, Z_0) = \emptyset$, since otherwise there is no vertex cover disjoint from Z_0. Let us partition the set $V \setminus Z$ into $V_1 = (V \setminus Z) \cap N(Z_0)$ and $V_0 = (V \setminus Z) \setminus V_1$. Less formally, we split the vertices of $V \setminus Z$ depending on whether they have a neighbour in Z_0 or not. Since we need to cover the edges adjacent to Z_0, any vertex cover disjoint from Z_0 contains all the vertices of V_1.

Observe that if there exists a vertex $v \in V_1$, such that $N(v) \subseteq Z_0$, no vertex cover disjoint from Z_0 is connected, since the vertex v can not be in the same connected component as any vertex of Z_1, meaning that this choice of Z_0 is invalid (see Fig. 1). Consequently each vertex in V_1 has at least one neighbour in Z_1. Moreover, $Z_1 \cup V_1$ forms a vertex cover of the graph G, as $V_0 \cup Z_0$ is an independent set. Hence we want to investigate how $Z_1 \cup V_1$ can be complemented with vertices of V_0, to make the vertex cover induce a connected subgraph. Let G' be the graph $G[Z_1 \cup V_0 \cup V_1]$ with connected components of $G[Z_1 \cup V_1]$ contracted to single vertices. Denote the set of vertices corresponding to contracted components of $G[Z_1 \cup V_1]$ as T. Note that G' is bipartite, since $G[V_0]$ is an independent set. By Lemma 7 we can find a minimum weight set $X \subseteq V_0$ of cardinality at most $(k - |Z_1| - |V_1|)$, such that $G'[T \cup X]$ is connected, which is equivalent to $G[Z_1 \cup V_1 \cup X]$ being connected. Observe that the size of the set T is upper bounded by the number of connected components of the induced subgraph $G[Z_1]$, as each vertex of V_1 has at least one neighbour in Z_1. Therefore, by Lemma 7, for a fixed choice of Z_1 we can find the set X in $O(2^{|T|}(k - |Z_1| - |V_1|)(|V(G')| + |E(G')|)) = O(2^{|\mathrm{cc}(G[Z_1])|}k(|V(G)| + |E(G)|))$ time and $O(2^{|T|}(k-|Z_1|-|V_1|)) = O(2^k k)$ space. Moreover for a uniform weight function, by Lemma 7, the running time is $O(2^{|\mathrm{cc}(G[Z_1])|}(|V(G)| + |E(G)|))$ and space usage is $O(2^k)$.

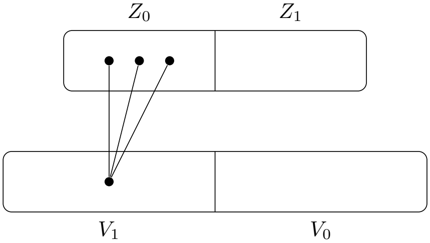


Fig. 1. An example of invalid choice of Z_0, since a vertex of V_1 has neighbours in Z_1

Summing up the running time over all the choices of Z_1, for which Z_0 is an independent set, by Lemma 6 applied to the graph $G[Z]$ we prove the total running time of our algorithm is $O(2^k k(|V(G)| + |E(G)|))$ for a general weight function and $O(2^k(|V(G)| + |E(G)|))$ for a uniform weight function. □

2.3 Combinatorial Bound

Now we prove Lemma 6, where we reduce the trivial $3^{|V|}$ bound to $3 \cdot 2^{|V|-1}$, by using a similar idea, as was previously used for Bandwidth [6,7] and Connected Vertex Cover [5].

Proof (of Lemma 6). Note, that we may rewrite the sum we want to bound as follows:

$$\sum_{\substack{V_1 \subseteq V \\ E(G[V \setminus V_1]) = \emptyset}} 2^{|\mathrm{cc}(G[V_1])|} = |\{(V_1, \mathcal{C}) : V_1 \subseteq V, \mathcal{C} \subseteq \mathrm{cc}(G[V_1]), E[G[V \setminus V_1]] = \emptyset\}| \,.$$

That is we count the number of pairs $(V_1, \mathcal{C})$, such that V_1 forms a vertex cover of G and $\mathcal{C}$ is any subset of connected components of the subgraph induced by V_1. Denote the set of all pairs $(V_1, \mathcal{C})$ we are counting as $\mathcal{S}$. Observe that we can easily construct an injection ϕ from $\mathcal{S}$ to $\{\mathbf{ii}, \mathbf{io}, \mathbf{o}\}^{|V|}$, where for a pair $(V_1, \mathcal{C})$ as $\phi((V_1, \mathcal{C}))(v)$ we set:

- **ii** (in-in) when $v \in V_1$ and the connected component of $G[V_1]$ containing v belongs to $\mathcal{C}$,
- **io** (in-out) when $v \in V_1$ and the connected component of $G[V_1]$ containing v does not belong to $\mathcal{C}$,
- **o** (out) when $v \notin V_1$.

Having any function $f : V \to \{\mathbf{ii}, \mathbf{io}, \mathbf{o}\}$, which belongs to the image of ϕ, we can reconstruct a pair $(V_1, \mathcal{C})$ (if it exists), such that $\phi((V_1, \mathcal{C})) = f$. However, the injection ϕ is not a surjection, for at least two reasons. Consider any $f \in \phi(\mathcal{S})$. Firstly, for any edge $uv \in E$, we have $f(u) \in \{\mathbf{ii}, \mathbf{io}\}$ or $f(v) \in \{\mathbf{ii}, \mathbf{io}\}$, since otherwise V_1 is not a vertex cover of G. Secondly, for any edge $uv \in E$, if we have $f(u) \in \{\mathbf{ii}, \mathbf{io}\}$, then either $f(v) = \mathbf{o}$ or $f(v) = f(u)$, because if both u and v belong to V_1, then they are part of exactly the same connected component C of $G[V_1]$, and therefore knowing $f(u)$ we can infer whether $C \in \mathcal{C}$ or $C \notin \mathcal{C}$.

Let us formalize the intuition above, to prove that for almost each vertex we have at most two, instead of three possibilities. Consider a spanning tree T of G and root it at an arbitrary vertex r. We construct the following function

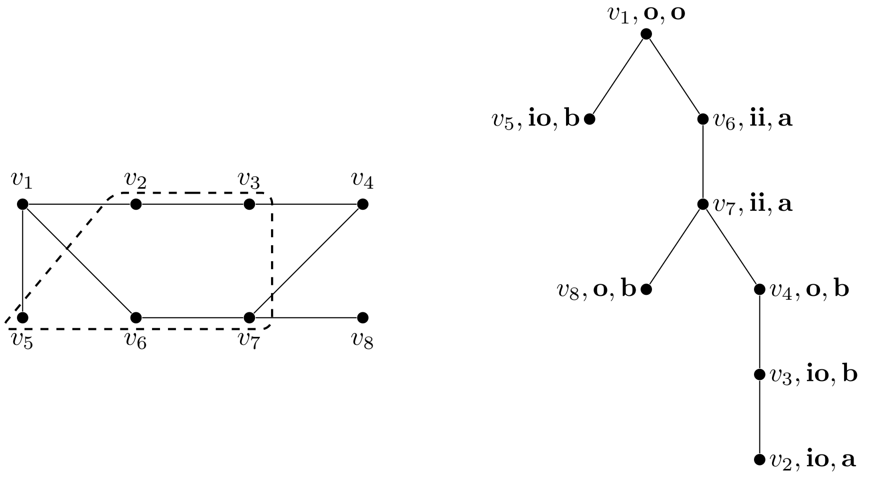

v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
$v_1, \mathbf{o}, \mathbf{o}$
$v_5, \mathbf{io}, \mathbf{b}$
$v_6, \mathbf{ii}, \mathbf{a}$
$v_7, \mathbf{ii}, \mathbf{a}$
$v_8, \mathbf{o}, \mathbf{b}$
$v_4, \mathbf{o}, \mathbf{b}$
$v_3, \mathbf{io}, \mathbf{b}$
$v_2, \mathbf{io}, \mathbf{a}$

Fig. 2. The set V_1 is enclosed within the dashed border, whereas $\mathrm{cc}(G[V_1]) = \{\{v_2, v_3\}, \{v_5\}, \{v_6, v_7\}\}$ and $\mathcal{C} = \{\{v_6, v_7\}\}$. On the right there is a tree T rooted at v_1, where for each vertex values assigned by $\phi((V_1, \mathcal{C}))$ and $\phi'((V_1, \mathcal{C}))$ are given. Note that the for the root both $\phi((V_1, \mathcal{C}))$ and $\phi'((V_1, \mathcal{C}))$ assign exactly the same value.

$\phi' : \mathcal{S} \to \{\mathbf{ii}, \mathbf{io}, \mathbf{o}\} \times \{\mathbf{a}, \mathbf{b}\}^{V \setminus \{r\}}$. For a given pair $(V_1, \mathcal{C}) \in \mathcal{S}$ we set $\phi'((V_1, \mathcal{C})) = (\phi((V_1, \mathcal{C}))(r), f)$, where the function $f : V \setminus \{r\} \to \{\mathbf{a}, \mathbf{b}\}$ is defined in a top-down manner, with respect to the tree T, as follows. Let $v \in V \setminus \{r\}$ and denote $p \in V$ as the parent of v in T.

- If $p \in V_1$, then if $v \in V_1$, we set $f(v) = \mathbf{a}$ and otherwise (if $v \notin V_1$), we set $f(v) = \mathbf{b}$.
- If $p \notin V_1$, then we have $v \in V_1$ (since otherwise V_1 would not be a vertex cover), and if the connected component of $G[V_1]$ containing v belongs to $\mathcal{C}$, then $f(v) = \mathbf{a}$, otherwise $f(v) = \mathbf{b}$.

Since ϕ' is also a surjection, we have $|\mathcal{S}| \leq 3 \cdot 2^{|V|-1}$, and the lemma follows. An example showing both functions ϕ, ϕ' is depicted in Fig. 2. □

2.4 Bipartite Steiner Tree

Here we prove Lemma 7, which concerns the following bipartite variant of the node-weighted Steiner tree problem.

Weighted Bipartite Steiner Tree
Input: An undirected bipartite graph $G = (V, E)$, a weight function $\omega : V \to \mathbb{R}_+$, an integer k and a set of terminals $T \subseteq V$, such that both T and $V \setminus T$ are independent sets in G.
Parameter: $|T|$
Goal: Find a minimum weight subset $X \subseteq V \setminus T$ of size at most k, such that $G[T \cup X]$ is connected.

Proof (of Lemma 7). By a dynamic programming routine, for each subset $T_0 \subseteq T$ and integer $0 \leq j \leq k$ we compute the value $t(T_0, j)$, defined as the minimum weight of a subset $X \subseteq V \setminus T$, satisfying:

- $|X| = j$,
- $N(X) = T_0$,
- $G[T_0 \cup X]$ is connected.

Less formally, the value $t(T_0, j)$ is the minimum weight of a set X of cardinality exactly j, such that $G[T_0 \cup X]$ induces a connected subgraph, and there is no edge from X to $T \setminus T_0$. Observe that $\min_{1 \leq j \leq k} t(T, j)$ is the minimum weight solution for the Weighted Bipartite Steiner Tree problem, therefore in the rest of the proof we describe how to compute all the $(k+1)2^{|T|}$ values t efficiently.

Initially for each $t_0 \in T$ we set $t(\{t_0\}, 0) := 0$, while all other values in the table t are set to ∞. Next, consider all the subsets $T_0 \subseteq T$ in the order of their increasing cardinality, and for each integer $0 \leq j < k$ and each vertex $v \in N(T_0)$ do

$$t(T_0 \cup N(v), j+1) := \min(t(T_0 \cup N(v), j+1), t(T_0, j) + \omega(v)) \, .$$

In the above expression we take the minimum between the current value and the solution that utilizes the vertex v. Note, that the assumption $v \in N(T_0)$ ensures, that vertices $N(v) \setminus T_0$ get connected to the vertices of T_0.

With this simple dynamic programming routine we compute all the values $t(T_0, j)$ in $O(2^{|T|}k(|V(G)| + |E(G)|)$ time and $O(2^{|T|}k)$ space. Note, that by standard methods we can reconstruct a set X corresponding to the value $t(T, j)$ in the same running time. Moreover, if the weight function is uniform, than the second dimension of our dynamic programming table is unnecessary, since the cardinality and weight of a set are equal. This observation reduces both the running time and space usage by a factor of k. □

3 Counting

In this Section we present a proof of Theorem 2, which is similar to the proof of Theorem 1.

Proof (of Theorem 2). Similarly as in the proof of Theorem 1, by using Lemma 5 in $O(2^k k(|V|+|E|))$ time we construct a set Z, which is a connected vertex cover of G of size at most k, or verify that such a set does not exist.

Next, we proceed as in the proof of Lemma 4, however we have to justify the assumption that G is a connected graph. When G contains at least two connected components containing at least two vertices each, then there is no connected vertex cover in the graph G. If there is one connected component containing at least two vertices, then no connected vertex cover contains any of the isolated vertices, hence we can remove them. Finally, when the graph contains only isolated vertices, then it admits an empty connected vertex cover and $|V|$ connected vertex covers containing a single vertex only.

The rest of the proof of Lemma 4 remains unchanged and what we are left with is to show an $O(2^{|T|}\text{poly}(|V|))$ running time algorithm for the following #Bipartite Steiner Tree problem.

#Bipartite Steiner Tree
Input: An undirected bipartite graph $G = (V, E)$, an integer k and a set of terminals $T \subseteq V$, such that both T and $V \setminus T$ are independent sets in G.
Parameter: $|T|$
Goal: Find the number of subsets $X \subseteq V \setminus T$ of size at most k, such that $G[T \cup X]$ is connected.

We do it similarly as Björklund et al. in Section 4 of [2], that is for each $T_0 \subseteq T$, $0 \leq j \leq k$ and $1 \leq c \leq |T_0|$ we define the value $t(T_0, c, j)$, which is equal to the number of subsets $X \subseteq V \setminus T$ of size exactly j, such that $N(X) \subseteq T_0$ and $G[T_0 \cup X]$ consists of exactly c connected components. To compute values of the table t one needs to use fast subset convolution and the details will be included in the full version of the paper. □

4 Polynomial Space

The only place in our algorithm, where we use exponential space is when solving the Bipartite Steiner Tree problem. If, instead of using Lemma 7 we use the algorithm of Nederlof [15], running in $O(2^{|T|}\text{poly}(n))$ time, we obtain an $O(2^k\text{poly}(n))$ time and polynomial space algorithm for the Connected Vertex Cover problem. The algorithm by Nederlof solves also the weighted case, but only when the weights are polynomially bounded integers, which is enough to prove Theorem 3. Unfortunately, we are not aware of an algorithm which counts the number of solutions to the Bipartite Steiner Tree problem in $2^{|T|}|V|^{O(1)}$ time and polynomial space (note that the algorithm of [15] counts the number of branching walks, not the number of subsets of vertices inducing a solution).

5 Conclusions and Open Problems

In [5] Cygan et al. have shown a randomized $O(3^k\text{poly}(n))$ algorithm for the Feedback Vertex Set problem, where we want to make the graph acyclic by removing at most k vertices. Is it possible to design a deterministic algorithm of the same running time?

The Cut&Count technique presented in [5] does not allow neither to count the number of solution nor to solve problems with arbitrary real weights. Nevertheless, for the Connected Vertex Cover problem we were able to solve both the weighted and counting variants in the same running time. Is it possible to design $O(c^{\text{tw}}\text{poly}(n))$ time algorithms for counting or weighted variants of the connectivity problems parameterized by treewidth for which the Cut&Count technique can be applied?

Finally, we know that it is not possible to count the number of connected vertex covers of size at most k in $O((2-\varepsilon)^k\text{poly}(n))$ time, unless SETH fails. Can we prove that we can not solve the decision version of the problem as well in such running time?

Acknowledgements. We thank Daniel Lokshtanov, Marcin Pilipczuk and Michał Pilipczuk for helpful discussions. Moreover we are grateful to the anonymous reviewers, especially for the comments regarding the counting variant of the problem.

References

1. Binkele-Raible, D.: Amortized Analysis of Exponential Time and Parameterized Algorithms: Measure and Conquer and Reference Search Trees. PhD thesis, University of Trier (2010)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Computing the tutte polynomial in vertex-exponential time. In: FOCS, pp. 677–686 (2008)
3. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. 411(40-42), 3736–3756 (2010)

4. Cygan, M., Dell, H., Lokshtanov, D., Marx, D., Nederlof, J., Okamoto, Y., Paturi, R., Saurabh, S., Wahlström, M.: On problems as hard as CNF-SAT. In: CCC (to appear, 2012)
5. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Ostrovsky, R. (ed.) FOCS, pp. 150–159. IEEE (2011)
6. Cygan, M., Pilipczuk, M.: Faster Exact Bandwidth. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 101–109. Springer, Heidelberg (2008)
7. Cygan, M., Pilipczuk, M.: Exact and approximate bandwidth. Theor. Comput. Sci. 411(40-42), 3701–3713 (2010)
8. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: On Multiway Cut Parameterized above Lower Bounds. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 1–12. Springer, Heidelberg (2012)
9. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through Colors and IDs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
10. Fernau, H., Manlove, D.: Vertex and edge covers with clustering properties: Complexity and algorithms. J. Discrete Algorithms 7(2), 149–167 (2009)
11. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica 20(4), 374–387 (1998)
12. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized Complexity of Generalized Vertex Cover Problems. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 36–48. Springer, Heidelberg (2005)
13. Mölle, D., Richter, S., Rossmanith, P.: Enumerate and Expand: Improved Algorithms for Connected Vertex Cover and Tree Cover. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 270–280. Springer, Heidelberg (2006)
14. Mölle, D., Richter, S., Rossmanith, P.: Enumerate and expand: Improved algorithms for connected vertex cover and tree cover. Theory Comput. Syst. 43(2), 234–253 (2008)
15. Nederlof, J.: Fast polynomial-space algorithms using mobius inversion: Improving on Steiner Tree and related problems. Algorithmica (page to appear)
16. Razgon, I., O'Sullivan, B.: Almost 2-SAT is fixed-parameter tractable. J. Comput. Syst. Sci. 75(8), 435–450 (2009)
17. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. 32(4), 299–301 (2004)
18. Ramanujan, M.S., Raman, V., Saurabh, S., Narayanaswamy, N.S.: LP can be a cure for parameterized problems. In: STACS, pp. 338–349 (2012)

Faster Parameterized Algorithms for Deletion to Split Graphs

Esha Ghosh[1], Sudeshna Kolay[1], Mrinal Kumar[2], Pranabendu Misra[3],
Fahad Panolan[1], Ashutosh Rai[1], and M.S. Ramanujan[1]

[1] The Institute of Mathematical Sciences, Chennai, India
{esha,skolay,fahad,ashutosh,msramanujan}@imsc.res.in
[2] Indian Institute of Technology, Madras
mrinalk@cse.iitm.ac.in
[3] Chennai Mathematical Institute
pranabendu@cmi.ac.in

Abstract. An undirected graph is said to be *split* if its vertex set can be partitioned into two sets such that the subgraph induced on one of them is a complete graph and the subgraph induced on the other is an independent set. We study the problem of deleting the minimum number of vertices or edges from a given input graph so that the resulting graph is split.We initiate a systematic study and give efficient fixed-parameter algorithms and polynomial sized kernels for the problem. More precisely,

1. for Split Vertex Deletion, the problem of determining whether there are k vertices whose deletion results in a split graph, we give an $\mathcal{O}^*(2^k)$[1] algorithm improving on the previous best bound of $\mathcal{O}^*(2.32^k)$. We also give an $\mathcal{O}(k^3)$-sized kernel for the problem.
2. For Split Edge Deletion, the problem of determining whether there are k edges whose deletion results in a split graph, we give an $\mathcal{O}^*(2^{O(\sqrt{k}\log k)})$ algorithm. We also prove the existence of an $\mathcal{O}(k^2)$ kernel.

In addition, we note that our algorithm for Split Edge Deletion adds to the small number of subexponential parameterized algorithms not obtained through bidimensionality, and on general graphs.

1 Introduction

The problem of editing (adding/deleting vertices/edges) to ensure that a graph has some property is a well studied problem in theory and applications of graph algorithms. When we want the resulting graph to be in a non-trivial hereditary (i.e. closed under induced subgraphs) graph class, the optimization versions of the corresponding vertex/edge deletion problems are known to be NP-complete by a classical result of Lewis and Yannakakis [14]. This problem has also been studied in generality under paradigms like approximation [9,16] and parameterized complexity [3,11]. When Π is a specific hereditary class like chordal or

[1] $\mathcal{O}^*()$ notation hides factors that are polynomial in the input size.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 107–118, 2012.

planar graphs, extensive work has been done to explore tight bounds [8,13,17,18]. In this paper, we initiate a study of these problems when Π is the class of all split graphs, which is also a hereditary graph class.

An undirected graph $G = (V, E)$ is said to be *split* if its vertex set V can be partitioned into two sets such that the induced subgraph on one of them is a complete graph and the induced subgraph on the other is an independent set. Split graphs were first studied by Földes and Hammer [7], and independently introduced by Tyshkevich and Chernyak [20]. In [7], the authors provided the following finite forbidden subgraph characterization of split graphs which gives us an easy polynomial time algorithm for recognizing split graphs.

Lemma 1. ([7]) *A graph is a split graph if and only if it contains no induced subgraph isomorphic to $2K_2$, C_4, or C_5. Here, K_2 is the complete graph on two vertices, C_i is a cycle on i vertices.*

In this paper, we study the following two problems.

Split Vertex Deletion (SVD)

Input:	Graph $G = (V, E)$, integer k
Parameter:	k
Question:	Does there exist a set of vertices of size at most k whose deletion from G results in a split graph?

Split Edge Deletion (SED)

Input:	Graph $G = (V, E)$, integer k
Parameter:	k
Question:	Does there exist a set of edges of size at most k whose deletion from G results in a split graph?

As the size of the forbidden set is finite, these problems become fixed-parameter tractable (see Section 2) due to a general result of Cai [3], when parameterized by k. One can also observe from Lemma 1, a fairly straightforward branching algorithm for both SVD and SED with running time $\mathcal{O}^*(5^k)$.

Recently, in [15,19], the authors obtained an $\mathcal{O}^*(2.32)^k$ algorithm for SVD by reducing the problem to the Above Guarantee Vertex Cover problem and using the fixed-parameter algorithm for it. In this paper, we improve this bound to $\mathcal{O}^*(2^k)$ by the combination of a bound on the number of split partitions of a split graph, and the well known technique of iterative compression. We also obtain an $\mathcal{O}(k^3)$ vertex kernel for the problem. Note that, this kernel is smaller than the kernel with $O(k^4)$ vertices, which can be obtained by an approach similar to d-Hitting Set [1]. We also observe that under certain complexity theoretic assumptions, we cannot obtain a subexponential algorithm for this problem. Indeed, the reduction from Vertex Cover, where we add a disjoint clique of size $k + 2$ to the given instance, along with the fact that Vertex Cover does not admit a subexponential algorithms unless the Exponential Time Hypothesis (ETH) fails [4], proves this observation.

For SED, we design a subexponential algorithm running in time $\mathcal{O}^*(2^{\mathcal{O}(\sqrt{k}\log k)})$ by combining the color and conquer approach [2], with the bound on the number of partitions of a split graph. This is probably the second problem (see [8]) having a subexponential algorithm on general graphs which doesn't use bidimensionality theory. We also revisit the kernelization algorithm for this problem given by Guo [11], and by using only a subset of the rules presented there, we prove a bound of $\mathcal{O}(k^2)$ vertices improving on Guo's bound of $\mathcal{O}(k^4)$. Furthermore, the Split Completion problem of adding at most k edges to a given graph to make it split, is equivalent to deleting at most k edges from the complement of the graph to make it split. Hence, the bound on the kernel and the subexponential algorithm which we prove for SED also holds for Split Completion. We also note that though computing a minimum split completion set is NP-complete, there is a linear time algorithm to compute a minimal split completion set [12].

2 Preliminaries

For a graph $G = (V, E)$, and a set A of edges, we denote by $V(A)$ the set of endpoints of the edges in A. For a set $S \subseteq V$, the *subgraph of* G *induced by* S is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{(u, v) \in E : u, v \in S\}$ and the subgraph obtained after deleting S is denoted as $G \setminus S$. Similarly, the *subgraph of* G *induced by an edge set* $A \subseteq E$ is defined as the subgraph of G with edge set A and vertex set $V(A)$. All vertices adjacent to a vertex v are called neighbours of v and the set of all such vertices is called the neighbourhood of v. Similarly, a non-adjacent vertex of v is called a non-neighbour and the set of all non-neighbours of v is called the non-neighbourhood of v. The neighborhood of v is denoted by $N(v)$. We say that v is *global* to a set Z if v is adjacent to all vertices of Z and we say that v is non-adjacent (or non-neighbor) to a set Z if v is not adjacent to any vertex of Z. For two sets X and Y, we say that X is *global* to Y if every vertex in X is global to Y and that X is non-adjacent to Y if every vertex in X is non-adjacent to Y.

Given a function $col : V \to C$ from the vertices of the graph G to a set of colors, C, we say that an edge $(u, v) \in E$ is monochromatic if $col(u) = col(v)$ and non-monochromatic otherwise.

A graph G is called a *split graph* if the vertex set V can be partitioned into two sets V_1 and V_2 such that $G[V_1]$ is a complete graph and $G[V_2]$ is an independent set. We call a set $S \subseteq V$ a *split vertex deletion* (svd) set if the graph $G[V \setminus S]$ is a split graph and a set $A \subseteq E$ is called a *split edge deletion* (sed) set if the graph $G[E \setminus A]$ is a split graph.

Definition 1. *Given a split graph $G = (V, E)$, a partition $(C \uplus I)$ of the vertex set into sets C and I is called a* split partition *of this split graph if $G[C]$ is a clique and $G[I]$ is an independent set.*

Given a split partition $(C_0 \uplus I_0)$ of a subgraph G' of a split graph G, we say that a split partition $(C \uplus I)$ of G is *consistent* with the partition $(C_0 \uplus I_0)$ if $C_0 \subseteq C$ and $I_0 \subseteq I$. We refer to an induced subgraph isomorphic to $2K_2$, or C_4 or C_5 as a *forbidden structure.*

Parameterized Complexity. For decision problems with input size n, and a parameter k, the goal in parameterized complexity is to design an algorithm with running time $f(k)n^{\mathcal{O}(1)}$ where f is a function of k alone, as contrasted with a $n^{k+\mathcal{O}(1)}$ algorithm. Problems which admit such algorithms are said to be fixed parameter tractable (FPT). We also call an algorithm with a running time of $f(k)n^{\mathcal{O}(1)}$, an FPT algorithm, and such a running time, an FPT running time. The theory of parameterized complexity was developed by Downey and Fellows [5]. For recent developments, see the book by Flum and Grohe [6].

Kernelization. A *kernelization* algorithm for a parameterized language L is a polynomial time procedure which takes as input an instance (x, k), where k is the parameter and returns an instance (x_1, k_1) such that $(x, k) \in L$ if and only if $(x_1, k_1) \in L$ and $|x_1| \leq h(k)$ and $k_1 \leq g(k)$, for some computable functions h, g. The returned instance is said to be the kernel for L.

3 An Improved Algorithm for Split Vertex Deletion

In this section, we present a faster parameterized algorithm for SVD by combining the technique of iterative compression along with a linear bound on the number of split partitions of split graphs. The following lemma is implied by Theorem 6.2, [10].

Lemma 2. (Theorem 6.2, [10]) *A split graph on n vertices can have at most $n + 1$ split partitions.*

We will now describe the application of the iterative compression technique to the SVD problem.

Iterative Compression for Split Vertex Deletion. Given an instance $(G = (V, E), k)$ of SVD, we let $V = \{v_1, \ldots, v_n\}$ and define vertex sets $V_i = \{v_1, \ldots, v_i\}$, and let the graph $G_i = G[V_i]$. We iterate through the instances (G_i, k) starting from $i = k + 3$. For the i^{th} instance, we try to find a solution $\hat{S}_i$ of size at most k, with the help of a *known* solution S_i of size at most $k + 1$. Formally, the compression problem we address is the following.

Split Vertex Deletion Compression (SVD Compression)

Input:	Graph $G = (V, E)$, an svd set $S \subseteq V$ of size at most $k + 1$, integer k
Parameter:	k
Question:	Does there exist an svd set of size at most k?

We reduce the SVD problem to $n - k - 2$ instances of the SVD Compression problem as follows. Let $I_i = (G_i, S_i, k)$ be the i^{th} instance. Clearly, the set V_{k+1} is a solution of size at most $k + 1$ for the instance I_{k+3}. It is also easy to see that if $\hat{S}_{i-1}$ is a solution of size at most k for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup \{v_i\}$ is a solution of size at most $k + 1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+3}, S_{k+3} = V_{k+1}, k)$ and look for a solution of size at most k for this instance. If there is such a solution $\hat{S}_{k+3}$, we set $S_{k+4} = \hat{S}_{k+3} \cup \{v_{k+4}\}$ and ask for a solution of size at most k for the instance I_{k+4} and so on. If, during any iteration, the corresponding instance does not have a solution of the required size, it implies that the original instance is also a No instance. This follows from the fact that if a graph G has a split vertex deletion set of size k, then any vertex induced subgraph of G also has a split vertex deletion set of size k. Finally, the solution for the original input instance will be $\hat{S}_n$. Since there can be at most n iterations, the total time taken to solve the original instance is bounded by n times the time required to solve the SVD Compression problem.

Our algorithm for SVD Compression is as follows. Let the input instance be $I = (G = (V, E), S, k)$. We guess a subset $Y \subseteq S$ with the intention of picking these vertices in our hypothetical solution for this instance and ignoring the rest of the vertices in S. We delete the set Y from the graph and decrease k appropriately. We then check if the graph $G[S \setminus Y]$ is a split graph and if it is not, then reject this guess of Y as a spurious guess. Suppose that $G[S \setminus Y]$ is indeed a split graph. We now guess and fix a split partition $(C_0 \uplus I_0)$ for this graph. By Lemma 2, we know that there are at most $k + 2$ such split partitions. The split partition we fix corresponds to the split partition *induced* by the hypothetical solution on the graph $G[S \setminus Y]$. Hence, it now remains to check if there is an SVD set of the appropriate size which is disjoint from $S \setminus Y$, and results in a split graph with a split partition *consistent* with $(C_0 \uplus I_0)$. More formally, we have an instance of the following problem.

Split Vertex Deletion Compression* (SVD Compression*)
Input: Graph $G = (V, E)$, an svd set $S \subset V$ such that $G[S]$ is a split graph, a split partition $(C_0 \uplus I_0)$ for the graph $G[S]$, integer k
Parameter: k
Question: Does there exist an svd set X of size at most k, disjoint from S such that $G \setminus X$ has a split partition consistent with $(C_0 \uplus I_0)$?

The following lemma gives a polynomial time algorithm for the above problem.

Lemma 3. [∗][2] Split Vertex Deletion Compression* *can be solved in time polynomial in the input size.*

Given Lemma 3, our algorithm for SVD Compression has a running time of $\mathcal{O}(\Sigma_{i=0}^{k}\binom{k+1}{i} \cdot k \cdot n^{\mathcal{O}(1)}) = \mathcal{O}^*(2^k)$, where the factor of k is due to the number of split partitions of $G[S \setminus Y]$ and $n^{\mathcal{O}(1)}$ is due to the time required to execute our algorithm for SVD Compression*.

Finally, since we solve at most n instances of SVD Compression, our algorithm for SVD runs in time $\mathcal{O}^*(2^k)$, giving us the following theorem.

Theorem 1. Split Vertex Deletion *can be solved in time $\mathcal{O}^*(2^k)$ time.*

[2] Proofs of results marked [∗] will appear in the full version of the paper.

3.1 A Cubic Kernel for Split Vertex Deletion

In this subsection, we use the structural claim made in the algorithm for SVD to design a vertex kernel of size $\mathcal{O}(k^3)$ for SVD. We design the kernel by introducing reduction rules which can be applied in polynomial time to reduce the instance. The reduction rules we present here are applied exhaustively and in the order in which they are presented.

We say that a reduction rule that is applied on an instance (G, k) to produce an instance (G', k') is *correct* if (G, k) is a YES instance if and only if (G', k') is a YES instance.

Reduction Rule 1. *Compute an inclusion wise maximal set of vertex disjoint forbidden structures greedily, and let the set of vertices involved in this set of forbidden structures be O^*. If $|O^*|$ exceeds $5k$, then return a trivial* No *instance.*

Moving forward, we assume that $|O^*| \leq 5k$. Note that $G \setminus O^*$ is a split graph, and let $(C^* \uplus I^*)$ be a split partition of this graph. Before we present the next reduction rule, we need the following definition.

Definition 2. *We say that a vertex v of G has a* high clique non-neighbourhood *if $|C^* \setminus N(v)| \geq k+2$. Similarly, v is said to have a* high independent set neighborhood *if $|I^* \cap N(v)| \geq k+2$.*

Let $H_i = \{x \in V : |C^* \setminus N(v)| \geq k+2\}$, and let $H_c = \{x \in V : |I^* \cap N(v)| \geq k+2\}$. It is easy to see that the vertices in H_i will either end up in the independent partition of the resulting split graph, or will get deleted and hence will be in the solution. Similarly for vertices in H_c. This justifies the correctness of the next reduction rule.

Reduction Rule 2. *If there is a vertex $v \in H_i \cap H_c$, then delete v and decrease k by 1.*

Lemma 4. [$*$] *Reduction Rule 2 is correct*

We now partition the vertex set of the resulting graph G as follows.

- Let $C_1 = H_c \cap C^*$ be the set of vertices of C^* which have high independent set neighborhood, and let $I_1 = H_i \cap I^*$, is the set of vertices of I^* which have high clique non-neighborhood.
- Similarly $C_o = H_c \cap O^*$, is the set of vertices of O^* which have high independent set neighborhood, and $I_o = H_i \cap O^*$, is the set of vertices of O^* which have high clique non-neighborhood.
- Let $C_1^* = C^* \setminus C_1$, and $I_1^* = I^* \setminus I_1$.

We first show that the size of at least one of the sets, C_1^* and I_1^* is bounded (by a linear function of k).

Lemma 5. [$*$] $|C_1^*| \leq 2k+2$ *or* $|I_1^*| \leq 2k+2$.

In order to describe the next reduction rule, we assume that $|I_1^*| \leq 2k+2$. Later, we give an analogous rule for the case when $|I_1^*| > 2k+2$ and $|C_1^*| \leq 2k+2$.

Our aim is to show that if $|C_1^*|$ is larger than ck^2 for some constant c, then some of the vertices can be deleted to get an equivalent instance where $|C_1^*|$ is at most ck^2.

Observe that, by definition, only I_1 and I_o have high non-neighborhoods to C^*. Let Y be the set of vertices in C_1^* which have a non-neighbour in $(O^* \setminus I_o) \cup I_1^*$. Then $|Y|$ is $O(k^2)$ as $(O^* \setminus I_o) \cup I_1^*$ has $O(k)$ vertices and each such vertex has at most $O(k)$ non-neighbours in C^*. The next reduction rule presents a way to bound the remaining vertices in C_1^*. Let $C_1^r = C_1^* \setminus Y$ (i.e. C_1^r is adjacent to all the vertices of $(O^* \setminus I_o) \cup I_1^*$.)

Reduction Rule 3. *If $|C_1^r| > k+2$, then delete all edges from C_1^r to $I_1 \cup I_o$ and delete all but $k+2$ vertices of C_1^r.*

The correctness of the rule follows from Lemma 6 below.

Lemma 6. [∗] *Reduction Rule 3 is correct.*

Now that we have bounded $|C_1^r|$ by $k+2$, we have the following lemma.

Lemma 7. *The number of vertices in C_1^* is $\mathcal{O}(k^2)$.*

The above rule has an analogous counterpart in the case when $|I_1^*| > 2k+2$ and $|C_1^*| \leq 2k+2$. The reduction rule and analysis are identical, except we now consider independent sets where we considered cliques and we consider neighbors where we considered non-neighbors. Hence, we simply state the reduction rule without proof.

Reduction Rule 4. *Consider the subset X of I_1^* which is non-adjacent to the sets $O^* \setminus C_o$ and C_1^*. If this set is larger than $k+2$, then truncate it to size $k+2$, that is remove all but $k+2$ vertices of X and make the remaining vertices global to all vertices in $C_1 \cup C_o$.*

In this case, we can prove a bound of $\mathcal{O}(k^2)$ on the size of the set I_1^* (analogous to Lemma 7). Combining the bounds resulting from the application of these two rules (Reduction Rules 3 and 4), we get the following lemma.

Lemma 8. *When none of the reduction rules presented thus far apply, the size of the set $C_1^* \cup I_1^*$ is bounded by $\mathcal{O}(k^2)$.*

Observe that the only unbounded sets at this point are C_1 and I_1. We will reduce these sets by devising rules similar to Reduction Rule 3.

Reduction Rule 5. *Consider the subset X of C_1 which is global to the sets $O^* \setminus I_o$ and I_1^*. If this set is larger than $k+2$, then remove edges from X to $I_1 \cup I_o$ and truncate it to size $k+2$.*

Similarly, the following rule reduces the size of I_1.

Reduction Rule 6. *Consider the subset X of I_1 which is non-adjacent to the sets $O^* \setminus C_o$ and C_1^*. If this set is larger than $k+2$, then truncate it to size $k+2$ and make it global to all vertices in $C_1 \cup C_o$.*

Similar to Lemma 7, we can bound the size of the set C_1 by k times the size of the set $(O^* \setminus I_o) \cup I_1^*$, and we can bound the size of the set I_1 by k times the size of the set $(O^* \setminus C_o) \cup C_1^*$. By using the bounds we have already proved for these sets, we get the following lemma.

Lemma 9. *When none of the reduction rules apply, the sets C_1 and I_1 contain $\mathcal{O}(k^3)$ vertices.*

Summing up the bounds we have obtained, leads to the following theorem.

Theorem 2. *There is a vertex kernel for* SVD *with $\mathcal{O}(k^3)$ vertices.*

4 An Improved Algorithm for SED

In this section, we present a subexponential algorithm for SED using the *Color and Conquer* approach introduced by Alon, Lokshtanov and Saurabh [2]. We first design a randomized subexponential algorithm for this problem which succeeds with high probability. We then describe a way of derandomizing this algorithm to obtain a deterministic algorithm.

4.1 A Randomized Subexponential Algorithm for SED

This algorithm consists of three steps. In the first step, we reduce the instance (G, k) to an equivalent instance (G', k') with $\mathcal{O}(k^2)$ vertices. In the second step, we color the vertices of the graph uniformly at random and we prove that with a sufficiently high probability, all the edges of some k-sized solution (if one exists) are non-monochromatic. Finally, we give an algorithm to check if a colored instance of SED has a non-monochromatic split edge deletion set of size at most k.

Kernelization. We first apply the kernelization algorithm (see Section 4.3) which, given an instance (G, k) of SED, in polynomial time, returns an equivalent instance (G', k') of SED such that the number of vertices in G' is $\mathcal{O}(k^2)$ and $k' \leq k$. In the rest of this section, we will assume that the given instance is an instance of this kind.

Probability of a Good Coloring. We now color the vertices of G independently and uniformly at random with $\sqrt{8k}$ colors and let A_c be the set of non-monochromatic edges. Suppose that $(G = (V, E), k)$ is a YES instance and let $A \subseteq E$ be a solution to this instance. We now show that the probability of A being contained in A_c is at least $2^{-\mathcal{O}(\sqrt{k})}$. We begin by estimating the probability of obtaining a proper coloring (making all the edges non-monochromatic) when applying the above random experiment on a graph with k edges.

Lemma 10. ([2]) *If the vertices of a graph on q edges are colored independently and uniformly at random with $\sqrt{8q}$ colors then the probability that G is properly colored is at least $(2e)^{-\sqrt{q/8}}$.*

Now, since we colored each vertex of the graph G independently, the graph induced on the set A, of size at most k, will be properly colored with probability at least $2^{-\mathcal{O}(\sqrt{k})}$, which gives us the following lemma.

Lemma 11. *Let $(G = (V, E), k)$ be a* Yes *instance of* SED *which is colored by the random process described above, and let $A \subset E$ be a solution for this instance. The probability that no edge in A is monochromatic is at least $2^{-\mathcal{O}(\sqrt{k})}$.*

Solving a Colored Instance. We now present an algorithm to test if there is a *colorful* (all edges non-monochromatic) split edge deletion set in a given colored instance of SED. In the colored instance, every vertex is colored with one of $\sqrt{8k}$ colors. We start with the following simple observation.

Observation 1 [∗] *Let $G = (V_1 \cup V_2 \cup ... \cup V_t, E)$ be a t-colored graph with color classes $V_1, \ldots, V_t$. If there exists a colorful split edge deletion set A in G, then $G[V_i]$ is a split graph for every V_i.*

We now proceed to the description of the algorithm. Suppose the given instance had a colorful split edge deletion set A. Observation 1 implies that $G[V_i]$ is a split graph and it remains a split graph in $G \setminus A$. Hence, we use Lemma 2 to enumerate the split partitions of $G[V_i]$ for each i. Fixing a split partition for each $G[V_i]$ results in a *combined* split partition for the vertices in V. There are $\mathcal{O}(k^2)$ split partitions for each V_i and $\mathcal{O}(\sqrt{k})$ such sets. Hence, there are $k^{\mathcal{O}(\sqrt{k})}$ combined split partitions. Now, it simply remains to check if there is a combined split partition $(C \uplus I)$ such that the number of edges in the graph $G[I]$ is at most k and return Yes if and only if there is such a combined split partition. Hence, we have the following lemma.

Lemma 12. *Given a colored instance (G, k) of* SED *of size $\mathcal{O}(k^2)$, we can test if there is a colorful SED set of size at most k in time $2^{\mathcal{O}(\sqrt{k} \log k)}$.*

Combining Lemmas 11 and 12, we get the following theorem.

Theorem 3. *There is a randomized FPT algorithm for* SED *running in time $2^{\mathcal{O}(\sqrt{k} \log k)} + n^{\mathcal{O}(1)}$ with a success probability of at least $2^{-\mathcal{O}(\sqrt{k})}$.*

4.2 Derandomization with Universal Coloring Families

For integers m, k and r, a family F of functions from $[m]$ to $[r]$ is called a universal (m, k, r)-coloring family if, for any graph G on the set of vertices $[m]$ with at most k edges, there exists a function $f \in F$ which gives a proper vertex coloring of G. Suppose the kernel we obtain has size bounded by ck^2, then an explicit construction of a $(ck^2, k, \sqrt{8k})$-coloring family is known to exist.

Theorem 4. ([2]) *There exists an explicit universal* $(ck^2, k, \sqrt{8k})$*-coloring family* F *of size at most* $2^{\mathcal{O}(\sqrt{k}\log k)}$.

Replacing the randomized coloring step of our algorithm with the universal coloring family given by Theorem 4, yields the following theorem.

Theorem 5. *There is an algorithm which solves* SED *in time* $2^{\mathcal{O}(\sqrt{k}\log k)} + n^{\mathcal{O}(1)}$.

4.3 Improved Kernel for SED

In this subsection, we use a subset of the reduction rules for SED given in [11] to show the existence of a kernel with a quadratic number of vertices. The following are the reduction rules which we will apply on the given instance.

Reduction Rule 1. ([11]) *Delete vertices from* G *which are not part of an induced subgraph isomorphic to* $2K_2$, C_4 *or* C_5.

From this point on, we refer to an induced subgraph isomorphic to $2K_2$, C_4 or C_5, as an induced $2K_2$, C_4 or C_5 respectively. When Reduction Rule 1 no longer applies, every vertex in G is part of some induced $2K_2$, C_4 or C_5.

Reduction Rule 2. ([11]) *If two adjacent edges* (u, v) *and* (u, w) *occur together in more than* k *induced* C_4*s, then delete* (u, v) *and* (u, w) *from* G *and add two edges* (a, v) *and* (b, w)*, where* a *and* b *are two new vertices of degree 1.*

Reduction Rule 3. ([11]) *If an edge* e *occurs in more than* k *induced* $2K_2$*'s, then delete* e *from* G *and reduce* k *by one.*

We refer to [11] for the correctness of the reduction rules. We apply the above reduction rules exhaustively, in the order in which they are presented, and obtain a reduced instance (G', k'). For the sake of notational convenience, we denote the reduced instance by (G, k). In the rest of this discussion, we will assume that the reduced instance is a YES instance and prove a bound on the size of the instance with this assumption. Let S be a minimal solution with at most k edges and let $(C \uplus I)$ be a split partition of the graph $G \setminus S$. We call a vertex of G *affected* if some edge in S is incident on it, and *unaffected* otherwise. Observe that there are at most $2k$ affected vertices in G. We now make the following important observation.

Observation 2 [∗] *All the affected vertices lie in the independent set* I.

Lemma 13. [∗] *(a) Every induced* C_4 *in* G *intersects* S *in exactly one edge, or in exactly two adjacent edges of* C_4 *or in all the four edges.*

(b) Every induced C_5 *in* G *intersects* S *in exactly two adjacent edges of* C_5 *or in exactly three adjacent edges of* C_5 *or in all the five edges.*

We now give a bound on the number of vertices in the set I, using the following lemmas.

Lemma 14. [∗] *There are $\mathcal{O}(k^2)$ vertices which are part of an induced $2K_2$ in G.*

Lemma 15. [∗] *If $v \in I$ is an unaffected vertex, then v is not part of an induced C_4 or C_5 in G.*

Since we have bounded the number of both affected and unaffected vertices in I, we have the following lemma.

Lemma 16. *There are $\mathcal{O}(k^2)$ vertices in the independent set I.*

We now proceed to bound the number of vertices inside the clique C. To do so, we introduce the notion of a *sliced* vertex. For every edge $(p, q) \in S$, and a vertex $v \in C$, we say that the edge (p, q) *slices* v if v is adjacent to p but not to q or vice versa. We say that a vertex $v \in C$ is *sliced* if some edge in S slices it and *unsliced* otherwise. Observe that the sets of sliced and unsliced vertices form a partition of C.

Lemma 17. [∗] *If $v \in C$ is not sliced by any edge in S, then v is not part of an induced C_4 or C_5 in G.*

Since every unsliced vertex must be part of an induced $2K_2$, by Lemma 14 the number of unsliced vertices in the set C is bounded by $\mathcal{O}(k^2)$. To bound the number of sliced vertices in C, we argue that each edge in S can slice $\mathcal{O}(k)$ vertices of C, resulting in the following lemma.

Lemma 18. [∗] *There are $\mathcal{O}(k^2)$ sliced vertices in C.*

We have thus bounded the number of vertices in C and I, and hence bounded the number of vertices of the graph G, leading to the following theorem.

Theorem 6. *There is a kernel for* SED *with $\mathcal{O}(k^2)$ vertices.*

5 Conclusion

In this paper we studied the parameterized complexity of deleting k edges/vertices to get to the class of split graphs. We obtained faster parameterized algorithms as well as smaller sized kernels for these problems. We leave open the following problems.

1. Can we get an $\mathcal{O}^*((2-\epsilon)^k)$ algorithm for Split Vertex Deletion? Or can we show that our $\mathcal{O}^*(2^k)$ algorithm is optimal under certain complexity theoretic assumptions?
2. Can we get an $\mathcal{O}^*(2^{O(\sqrt{k})})$ algorithm for Split Edge Deletion?

It will also be interesting to get improved kernels for these problems. Finally, an interesting project could be to identify other parameterized problems that admit subexponential algorithms on general graphs.

Acknowledgements. We wish to thank Venkatesh Raman and Saket Saurabh for the insightful discussions which led to this work, and the anonymous reviewers for the numerous comments towards improving the presentation of the paper.

References

1. Abu-Khzam, F.: A kernelization algorithm for d-hitting set. Journal of Computer and System Sciences 76(7), 524–531 (2010)
2. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 49–58. Springer, Heidelberg (2009)
3. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processing Letters 58(4), 171–176 (1996)
4. Cai, L., Juedes, D.: On the existence of subexponential parameterized algorithms. J. Comput. Syst. Sci. 67, 789–807 (2003)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2006)
7. Foldes, S., Hammer, P.: Split graphs. Congressus Numerantium 19, 311–315 (1977)
8. Fomin, F.V., Villanger, Y.: Subexponential parameterized algorithm for minimum fill-in. In: SODA, pp. 1737–1746 (2012)
9. Fujito, T.: A unified approximation algorithm for node-deletion problems. Discrete Appl. Math. 86, 213–231 (1998)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
11. Guo, J.: Problem Kernels for NP-Complete Edge Deletion Problems: Split and Related Graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 915–926. Springer, Heidelberg (2007)
12. Heggernes, P., Mancini, F.: Minimal split completions. Discrete Applied Mathematics 157(12), 2659–2669 (2009)
13. Heggernes, P., van't Hof, P., Jansen, B.M.P., Kratsch, S., Villanger, Y.: Parameterized Complexity of Vertex Deletion into Perfect Graph Classes. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 240–251. Springer, Heidelberg (2011)
14. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. J. Comput. Syst. Sci. 20(2), 219–230 (1980)
15. Lokshtanov, D., Narayanaswamy, N.S., Raman, V., Ramanujan, M.S., Saurabh, S.: Faster parameterized algorithms using linear programming. CoRR, abs/1203.0833 (2012)
16. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. ACM 41, 960–981 (1994)
17. Marx, D.: Chordal deletion is fixed-parameter tractable. Algorithmica 57(4), 747–768 (2010)
18. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. Algorithmica 62(3-4), 807–822 (2012)
19. Narayanaswamy, N.S., Raman, V., Ramanujan, M.S., Saurabh, S.: Lp can be a cure for parameterized problems. In: STACS, pp. 338–349 (2012)
20. Tyshkevich, R.I., Chernyak, A.A.: Yet another method of enumerating unmarked combinatorial objects. Mathematical Notes 48, 1239–1245 (1990)

A Single-Exponential FPT Algorithm for the K_4-Minor Cover Problem*

Eun Jung Kim[1], Christophe Paul[2], and Geevarghese Philip[3]

[1] CNRS, LAMSADE, Paris, France
eunjungkim78@gmail.com
[2] CNRS, LIRMM, Montpellier, France
paul@lirmm.fr
[3] MPII, Saarbrücken, Germany
gphilip@mpi-inf.mpg.de

Abstract. Given an input graph G on n vertices and an integer k, the parameterized K_4-MINOR COVER problem asks whether there is a set S of at most k vertices whose deletion results in a K_4-minor free graph or, equivalently, in a graph of treewidth at most 2. The problem can thus also be called TREEWIDTH-2 VERTEX DELETION. This problem is inspired by two well-studied parameterized vertex deletion problems, VERTEX COVER and FEEDBACK VERTEX SET, which can be expressed as TREEWIDTH-t VERTEX DELETION problems: $t = 0$ for VERTEX COVER and $t = 1$ for FEEDBACK VERTEX SET. While a single-exponential FPT algorithm has been known for a long time for VERTEX COVER, such an algorithm for FEEDBACK VERTEX SET was devised comparatively recently. While it is known to be unlikely that TREEWIDTH-t VERTEX DELETION can be solved in time $c^{o(k)} \cdot n^{O(1)}$, it was open whether the K_4-MINOR COVER could be solved in single-exponential FPT time, i.e. in $c^k \cdot n^{O(1)}$ time. This paper answers this question in the affirmative.

1 Introduction

Given a set $\mathcal{F}$ of graphs, the parameterized $\mathcal{F}$-MINOR COVER problem is to identify a set S of at most k vertices — if it exists — in an input graph G such that the deletion of S results in a graph which does not have any graph from $\mathcal{F}$ as a minor; the parameter is k. Such a set S is called an *$\mathcal{F}$-minor cover* (or an *$\mathcal{F}$-hitting set*) of G. A number of fundamental graph problems can be viewed as $\mathcal{F}$-MINOR COVER problems. Well-known examples include VERTEX COVER ($\mathcal{F} = \{K_2\}$), FEEDBACK VERTEX SET ($\mathcal{F} = \{K_3\}$), and more generally TREEWIDTH-t VERTEX DELETION for any constant t, which asks whether an input graph can be converted to one with treewidth at most t by deleting at most k vertices. Observe that for $t = 0$ and 1, TREEWIDTH-t VERTEX DELETION is equivalent to VERTEX COVER and FEEDBACK VERTEX SET, respectively. In addition to its theoretical importance, the TREEWIDTH-t VERTEX DELETION problem has important practical applications as well. For example, even for

* This work is supported by the ANR project AGAPE (ANR-09-BLAN-0159).

small values of t, efficient algorithms for this problem would improve algorithms for inference in Bayesian Networks as a subroutine of the *cutset conditioning method* [1]. This method is practical only for small values of t, and therefore efficient algorithms for small treewidth t are desirable.

In this paper we consider the parameterized $\mathcal{F}$-MINOR COVER problem for $\mathcal{F} = \{K_4\}$, which is equivalent to TREEWIDTH-2 VERTEX DELETION. The NP-hardness of this problem follows from a classical result [2]. Its fixed-parameter tractability—that the problem can be solved in time $f(k) \cdot n^{O(1)}$ for an input graph on n vertices, where f is a computable function—follows from two celebrated meta-results: the Graph Minor Theorem of Robertson and Seymour [3] and Courcelle's theorem [4]. Unfortunately, the resulting algorithms involve huge exponential functions in k and are impractical even for small values of k.

For TREEWIDTH-t VERTEX DELETION, *single-exponential* parameterized algorithms—those which run in $c^k \cdot n^{O(1)}$ time on an input graph on n vertices, where c is a constant—are known only for $t = 0$ and $t = 1$. Indeed, for $t = 0$ (VERTEX COVER), a simple $O(2^k \cdot n)$-time algorithm is an oft-quoted first example for a parameterized algorithm [5–7]. For $t = 1$ (FEEDBACK VERTEX SET), no single-exponential algorithm was known for many years until Guo *et al.* [8] and Dehne *et al.* [9] independently discovered such algorithms. The fastest known deterministic algorithm for this problem runs in time $O(3.83^k \cdot n^2)$ [10]. The fastest known *randomized* algorithm, developed by Cygan et al., runs in $O(3^k \cdot n^{O(1)})$ time [11]. Very recently, Fomin et al. [12] presented $2^{O(k \log k)} \cdot n^{O(1)}$-time algorithms for TREEWIDTH-t VERTEX DELETION, for any constant t. In this paper we prove the following result for $t = 2$:

Theorem 1. *The K_4-*MINOR COVER *problem can be solved in $2^{O(k)} \cdot n^{O(1)}$ time.*

Our single-exponential parameterized algorithm for K_4-minor cover is based on iterated compression [13]. This allows us to focus on the *disjoint version* of the K_4-minor cover problem: given a solution S, find a smaller solution disjoint from S. We employ a search tree method to solve the disjoint problem. Although our algorithm shares the spirit of Chen et al.'s elegant iterated compression algorithm for FEEDBACK VERTEX SET [14], our need to cover K_4-minors—instead of K_3-minors—results in a compression step which is quite involved. To bound the branching degree by a constant, we make use of three key factors. First, we employ protrusion replacement, a technique developed to establish a meta theorem for polynomial-size kernels [15–17], with modifications that are needed to facilitate its use in the context of iterated compression. Second, we introduce the notion of an *extended* SP-decomposition for treewidth-two graphs, which makes it easier to exploit their structure. Finally, the running time analysis makes use of properties of the extended SP-decomposition and a measure which keeps track of the biconnectivity of a part of the graph.

2 Notation and Preliminaries

We follow the graph terminology of Diestel's textbook [18]. Our graphs are undirected, loopless and may contain parallel edges. We use "biconnected" as a synonym for “2-connected”. For a vertex set X in a graph $G = (V, E)$, the *boundary* $\partial_G(X)$ of X is the set $N(V \setminus X)$, i.e. the set of vertices in X which are adjacent with at least one vertex in $V \setminus X$. We sometimes omit the subscript when it is clear from the context. We use $tw(G)$ to denote the treewidth of a graph G.

When a graph H is a topological minor of a graph G as witnessed by a subgraph G' of G, we say that G' is an *H-subdivision* in G. In an H-subdivision G' of G, the vertices which correspond to the original vertices of H are called the *branching nodes*; the other vertices of G' are called the *subdividing nodes*. If the maximum degree of H is at most three, then G contains H as a minor if and only if it contains H as a topological minor [18]. A *θ_3-subdivision* is a graph which consists of three vertex disjoint paths between two branching vertices, called its *poles*.

Series-parallel graphs are a special class of graphs with a simple structure, which can be constructed by starting from the single edge and recursively applying the so-called series and parallel compositions [19]. This recursive construction can be represented by a so-called *SP-tree*, a canonical form of which can be constructed in linear time in the size of the series-parallel graph [20]. For any graph G, $tw(G) \leq 2$ if and only if every block of G is a series-parallel graph [20,21].

Extended SP-Decompositon. The *block tree* $\mathcal{B}_G$ of a connected graph G has a node set consisting of all the blocks and cut vertices of G, and a block B and a cut vertex c are adjacent in $\mathcal{B}_G$ if and only if B contains c. We now define the notion of an *extended SP-decomposition* of a connected graph of treewidth at most two. In general, an extended SP-decomposition of a graph is a collection of extended SP-decompositions of its connected components.

Let $\mathcal{B}_G$ be the block tree of a treewidth-two graph G. We fix an arbitrary cut node c_{root} of $\mathcal{B}_G$. The *oriented block tree* $\boldsymbol{\mathcal{B}}_G$ is obtained by orienting the edges of $\mathcal{B}_G$ outward from c_{root}. If $\mathcal{B}_G$ consists of a single node, it is regarded as an oriented block tree by itself. An *extended SP-decomposition* of G is a pair $(T, \mathcal{X} = \{X_\alpha : \alpha \in V(T)\})$, where T is a rooted tree whose vertices are called *nodes* and $\mathcal{X} = \{X_\alpha : \ \alpha \in V(T)\}$ is a collection of subsets of $V(G)$, one for each node in T. We say that X_α is the *label* of node α.

- For each block B of G, let $(T^B, \mathcal{X}^B)$ be a (canonical) SP-tree of $G[B]$ such that $c(B)$ is one of the terminals associated to the root node of T^B. A leaf node of T^B is called an *edge node*.
- For each cut vertex c of G, add to $(T, \mathcal{X})$ a *cut node* α with $X_\alpha = \{c\}$.
- For each block B of G, let the root node of $(T^B, \mathcal{X}^B)$ be a child of the unique cut node α in T which satisfies $X_\alpha = \{c(B)\}$.
- For a cut vertex c of G, let $B = B(c)$ be the unique block such that $(B, c) \in E(\boldsymbol{\mathcal{B}}_G)$. Let β be an arbitrary leaf node of the (canonical) SP-tree $(T^B, \mathcal{X}^B)$ such that $c \in X_\beta$ (note that such a node always exists). Make the cut node α of $(T, \mathcal{X})$ labeled by $\{c\}$ a child of the leaf node β.

Let α be a node of T. Then T_α is the subtree of T rooted at node α; E_α is the set of edges $(u, v) \in E(G)$ such that there exists an edge node $\alpha' \in V(T_\alpha)$

with $X_{\alpha'} = \{u, v\}$; and G_α is the — not necessarily induced — subgraph of G with the vertex set $V_\alpha := \bigcup_{\alpha' \in V(T_\alpha)} X_{\alpha'}$ and the edge set E_α. We define $Y_\alpha := V_\alpha \setminus X_\alpha$. We say that a node α of $(T, \mathcal{X})$ which is not a cut node is *inherited from* $(T^B, \mathcal{X}^B)$, where B is the block to which α belongs. Let α be inherited from $(T^B, \mathcal{X}^B)$. We use T^B_α to denote the SP-tree naturally associated with the subtree of T^B rooted at α. By G^B_α we denote the SP-graph represented by the SP-tree T^B_α, where $(T^B, \mathcal{X}^B)$ inherits α. The vertex set of G^B_α is denoted V^B_α. Note that for every node α, G_α is connected and that $\partial_G(V_\alpha) \subseteq X_\alpha$.

Soundness proofs and running time bounds for all the reduction rules, and proofs of those statements which are labelled with a $\star$, can be found in the full version of this paper [22].

3 The Algorithm

Our algorithm for K_4-minor cover uses various techniques from parameterized complexity. First, an *iterated compression* [13] step reduces K_4-minor cover to the so-called DISJOINT K_4-MINOR COVER problem, where in addition to the input graph we are given a solution set to be improved. Then a BRANCH-OR-REDUCE process develops a *bounded search tree*. We start with the definition of the compression problem for K_4-MINOR COVER.

Given a subset S of vertices, a K_4-minor cover W of G is S*-disjoint* if $W \cap S = \emptyset$. We omit the mention of S when it is clear from the context. If $|W| \leq k - 1$, then we say that W is *small*.

DISJOINT K_4-MINOR COVER PROBLEM

Input:	A graph G and a K_4-minor cover S of G
Parameter:	The integer $k \leq \lvert S\rvert$
Output:	A small S-disjoint K_4-minor cover W of G, if one exists. Otherwise return NO.

To prove Theorem 1, it is sufficient to show that DISJOINT K_4-MINOR COVER can be solved in $2^{O(k)} \cdot n^{O(1)}$ time; this follows by an argument which is now standard in the context of iterated compression [14,23,24]. Observe that both $G[V \setminus S]$ and $G[S]$ are K_4-minor-free. Indeed if $G[S]$ is not K_4-minor-free, then the answer to DISJOINT K_4-MINOR COVER is trivially NO.

3.1 A Disjoint Protrusion Reduction Rule

A subset X of the vertex set of a graph G is a t*-protrusion* of G if $tw(G[X]) \leq t$ and $|\partial(X)| \leq t$. Our algorithm uses a modified version of the "protrusion reduction" technique [15,16]. The adaptation is required because we have to apply the technique to the "disjoint" version of the problem. In essence, our (adapted) protrusion lemma for disjoint parameterized problems says that a 'large' protrusion which is disjoint from the forbidden set S can be replaced by a 'small' protrusion which is again disjoint from S.

Reduction Rule 1. (Generic disjoint protrusion rule) *Let (G, S, k) be an instance of* DISJOINT K_4-MINOR COVER *and X be a t-protrusion such that $X \cap S = \emptyset$. Then there exists a computable function $\gamma(.)$ and an algorithm which computes an equivalent instance (G', S, k') in time $O(|X|)$ such that $G[S]$ and $G'[S]$ are isomorphic, $G' - S$ is K_4-minor-free, $|V(G')| < |V(G)|$ and $k' \leq k$, provided $|X| > \gamma(2t + 1)$.*

We remark that some of the reduction rules in the next subsection are instantiations of the generic disjoint protrusion rule. However, to ease the algorithm analysis, the generic rule above is used only on t-protrusions whose boundary size is 3 or 4. For protrusions with boundary size 1 or 2, we shall instead apply the following explicit reduction rules.

3.2 Explicit Reduction Rules

Let F denote the subset $V(G) \setminus S$ of vertices. For a vertex $v \in F$, let $N_S(v)$ denote the neighbours of v which belong to S. By $N_i \subseteq F$ we refer to the set of vertices v in F with $|N_S(v)| = i$. In each of the next three rules, S and k are unchanged ($S' = S$, $k' = k$).

Reduction Rule 2. (1-boundary rule) *Let X be a subset of F.* **(a)** *If $G[X]$ is a connected component of G or of $G \setminus e$ for some bridge e, then delete X.* **(b)** *If $|\partial_G(X)| = 1$, then delete $X \setminus \partial_G(X)$.*

Reduction Rule 3. (Bypassing rule) *Bypass every vertex v of degree two in G with neighbours $u_1 \in V$, $u_2 \in F$. That is, delete v and its incident edges, and add the new edge (u_1, u_2).*

Reduction Rule 4. (Parallel rule) *If there is more than one edge between $u \in V$ and $v \in F$, then delete all these edges except for one.*

The next two reduction rules are somewhat more technical, and their proofs of correctness require a careful analysis of the structure of K_4-subdivisions.

Reduction Rule 5. (Chandelier rule) *Let $X = \{u_1, \ldots, u_\ell\}$ be a subset of F, and let x be a vertex in S such that $G[X]$ contains the path $u_1, \ldots, u_\ell$, $N_S(u_i) = \{x\}$ for every $i = 1, \ldots, \ell$, and vertices $u_2, \ldots, u_{\ell-1}$ have degree exactly 3 in G. If $\ell \geq 4$, contract the edge $e = (u_2, u_3)$ (and apply Rule 4 to remove the parallel edges created).*

The intuition behind the correctness of the Chandelier rule is that such a set X cannot host all the four branching nodes of a K_4-subdivision. Our last reduction rule is an explicit 2-protrusion rule. In the particular case when the boundary size is exactly two, the candidate protrusions for replacement are either a single edge or a θ_3 (see Figure 1).

Reduction Rule 6. (2-boundary rule) *Let $X \subseteq F$ be such that $G[X]$ is connected, $\partial(X) = \{s, t\}$ (and thus, $X \setminus \{s, t\} \subseteq N_0$). Then we do the following. (1) Delete $X \setminus \{s, t\}$. (2) If $G[X] + (s, t)$ is a series parallel graph and $|X| > 2$, then add the edge (s, t) (if it is not present). Else if $G[X] + (s, t)$ is not a series parallel graph and $|X| > 4$, add two new vertices a, b and the edges $\{(a, b), (a, t), (a, s), (b, t), (b, s)\}$ (see Figure 1).*

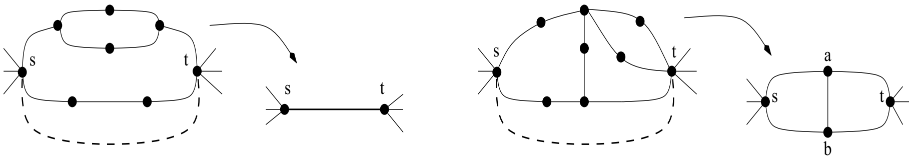


Fig. 1. If $G[X] + (s, t)$ is an SP-graph, we can safely replace $G[X]$ by the edge (s, t). Otherwise $G[X]$ can be replaced by a subdivision of θ_3 with poles a and b in which s and t are subdividing nodes.

We say that an instance of DISJOINT K_4-MINOR COVER is *reduced* if none of the reduction rules 2 - 6 results in changes to the instance.

3.3 Branching Rules

A *branching rule* is an algorithm which, given an instance (G, S, k), outputs a set of d instances $(G_1, S_1, k_1) \ldots (G_d, S_d, k_d)$ for some constant $d > 1$ (d is the *branching degree*). A branching rule is *safe* if (G, S, k) is a YES-instance if and only if there exists i, $1 \leq i \leq d$, such that (G_i, S_i, k_i) is a YES instance. We now present three generic branching rules, with potentially unbounded branching degrees. Later we describe how to apply these rules so as to bound the branching degree by a constant. Given a vertex $s \in S$, we denote by $cc_S(s)$ the connected component of $G[S]$ which contains s. Likewise, $bc_S(s)$ denotes the biconnected component of $G[S]$ containing s. It is easy to see that three branching rules below are safe.

Branching Rule 1. *Let (G, S, k) be an instance of* DISJOINT K_4-MINOR COVER *and let X be a subset of F such that $G[S \cup X]$ contains a K_4-subdivision. Then branch into the instances $(G - \{x\}, S, k - 1)$ for every $x \in X$.*

Branching Rule 2. *Let (G, S, k) be an instance of* DISJOINT K_4-MINOR COVER *and let X be a connected subset of F. If S contains two vertices s_1 and s_2, each of which has a neighbour in X, and such that $cc_S(s_1) \neq cc_S(s_2)$, then branch into the instances*

- $(G - \{x\}, S, k - 1)$ *for every* $x \in X$
- $(G, S \cup X, k)$

Branching Rule 3. *Let (G, S, k) be an instance of* DISJOINT K_4-MINOR COVER *and let X be a connected subset of F. If S contains two vertices s_1 and s_2, each of which has a neighbour in X, and such that $cc_S(s_1) = cc_S(s_2)$ and $bc_S(s_1) \neq bc_S(s_2)$, then branch into the instances*

– $(G - \{x\}, S, k - 1)$ *for every* $x \in X$
– $(G, S \cup X, k)$

We shall apply branching rule 1 in three different cases: (i) X is a singleton $\{x\}$ for $x \in F$, (ii) X is connected, or (iii) X consists of a pair of non-adjacent vertices of F. An instance (G, S, k) is said to be a *simplified instance* if it is a reduced instance and if none of the branching rules 1–3 applies on singleton sets $X = \{v\}$, for any $v \in F$. Such an instance has a useful property.

Lemma 1. [$\star$] *If* (G, S, k) *is a simplified instance of* DISJOINT K_4-MINOR COVER*, then* $F = N_0 \cup N_1 \cup N_2$*.*

An instance (G, S, k) of DISJOINT K_4-MINOR COVER is *independent* if (a) F is an independent set; (b) every vertex of F belongs to N_2; (c) the two neighbours of every vertex of F belong to the same biconnected component of $G[S]$, and (d) $G[S \cup \{x\}]$ is K_4-minor free for every $x \in F$. The instance becomes independent if case (ii) of branching rule 1 is applied exhaustively.

Theorem 2. [$\star$] *Let* (G, S, k) *be an instance of* DISJOINT K_4-MINOR COVER*. If (i) none of the reduction rules applies and (ii) no branching rules applies on connected subsets* $X \subseteq F$*, then* (G, S, k) *is an independent instance.*

We now construct an auxiliary graph $G^*(S)$: its vertex set is F, and (u, v) is an edge in $G^*(S)$ if and only if $G[S \cup \{u, v\}]$ contains K_4 as a minor. The next theorem says that to solve DISJOINT K_4-MINOR COVER on the independent instance, it is sufficient to exhaustively apply case (iii) of branching rule 1.

Theorem 3. [$\star$] *Let* (G, S, k) *be an independent instance of* DISJOINT K_4-MINOR COVER*. Then* $W \subseteq F$ *is a disjoint* K_4*-minor cover of* G *if and only if it is a vertex cover of* $G^*(S)$*.*

3.4 The Algorithm and Its Complexity Analysis

We now present the algorithm. At each node of the computation tree associated with a given instance (G, S, k), one of the following operations is performed. As each operation either returns a solution (as in (a),(e)) or generates a set of instances (as in (b)-(d)), the overall application of the operations can be depicted as a search tree.

(a) if $(k < 0)$ or $(k \leq 0, tw(G) > 2)$ or $(tw(G[S]) > 2)$, then return NO;
(b) if the instance is not reduced, apply one of the reduction rules 2–5. If none of these applies, then apply reduction rule 6;
(c) if the instance is not simplified, apply one of branching rules 1–3 on the singleton sets $\{x\}$ for each $x \in F$;
(d) if the instance is simplified, apply the procedure BRANCH-OR-REDUCE;
(e) if the application of BRANCH-OR-REDUCE marks every node of $(T, \mathcal{X})$, then the instance is an independent instance; solve it in $2^k \cdot n^{O(1)}$ time (Theorem 3).

The procedure Branch-or-reduce works in a bottom-up manner on an extended SP-decomposition $(T, \mathcal{X})$ of $G[F]$. Initially the nodes of $(T, \mathcal{X})$ are unmarked. Starting from a lowest node, Branch-or-reduce recursively tests if we can apply one of the branching rules on a subgraph associated with a lowest unmarked node. If the branching rules do not apply, the procedure tries to detect a large protrusion (see Lemma 3) and to reduce the instance using the generic protrusion rule (reduction rule 1). Once either a branching rule or the protrusion rule has been applied, the procedure Branch-or-reduce terminates. The output is a set of instances of disjoint K_4-minor cover, possibly a singleton.

A key part of the complexity analysis of the algorithm consists of showing that the sets X on which branching rules 1–3 are applied, in lines 4, 7, and 10, are of size bounded by some constant. This is done by a series of technical lemmas. To simplify the notation, in the following we use G_α instead of $G[F]_\alpha$ for a node α of T. Similarly, we use the names V_α, $Y_\alpha = V_\alpha \setminus X_\alpha$ and V_α^B to denote the various named subsets of $V(G[F]_\alpha)$.

Lemma 2. [$\star$] *Let W and Z be disjoint vertex subsets of a graph G such that $G[W]$ is biconnected, $G[Z]$ is connected and $|N_W(Z)| \geq 3$. Then $G[W \cup Z]$ contains a K_4-subdivision.*

Lemma 3. [$\star$] *Let (G, S, k) be a simplified instance and let α be a lowest node of the extended SP-decomposition $(T, \mathcal{X})$ of $G[F]$ which is considered at line 11 of Algorithm 1. If α is a P-node inherited from the SP-tree of block B, then $|\partial_G(V_\alpha^B) \setminus X_\alpha| \leq 2$ and V_α^B is a 4-protrusion.*

The next two lemmas show that applying Branch-or-reduce in a bottom-up manner enables us to bound the branching degree of the Branch-or-reduce procedure. Lemma 4 states that for every marked node α, the graph G_α is of constant-size.

Lemma 4. [$\star$] *Let (G, S, k) be a simplified instance of* disjoint K_4-minor cover *and let α be a marked node of the extended SP-decomposition $(\mathcal{X}, T)$ of $G[F]$. Then $|V_\alpha| \leq c_1 := 12(\gamma(8) + 2c_0)$.*

Lemma 5. [$\star$] *Let (G, S, k) be a simplified instance of* disjoint K_4-minor cover *and let α be a lowest unmarked node of $(T, \mathcal{X})$ of $G[F]$. In polynomial time, one can find*

(a) a path X of size at most $2c_1$ satisfying the conditions of line 3 (resp. line 6), if the test at line 2 (resp. 5) succeeds*;*
(b) a subset $X \subseteq V_\alpha$ of size bounded by $2c_1$ satisfying the condition of line 9, if the test at line 8 succeeds*;*

For analysing the running time of our algorithm, we introduce the following measure

$$\mu := (2c_1 + 2)k + (2c_1 + 2)\#cc(G[S]) + \#bc(G[S])$$

where $\#cc(G[S])$ (resp. $\#bc(G[S])$) denotes the number of connected components (resp. biconnected components) of $G[S]$.

Algorithm 1. Branch-or-reduce

```
Input: A simplified instance $(G, S, k)$ of disjoint $K_4$-minor cover, together
       with an extended SP-decomposition $(T, \mathcal{X})$ of $G[F]$.
Output: A set of instances of disjoint $K_4$-minor cover.
while $T$ contains unmarked nodes do
    Let $\alpha$ be an unmarked node at the farthest distance from the root of $T$;
    if $S$ contains two vertices $x_u \in N_S(u)$ and $x_v \in N_S(v)$ with $u, v \in V_\alpha$ and
    $cc_S(x_u) \neq cc_S(x_v)$ then
        Let $X$ be a path in $G_\alpha$ between two such vertices $u$ and $v$ such that
        $X \setminus \{u, v\} \subseteq N_0$;
        Apply branching rule 2 to $X$; terminate;
    if $S$ contains two vertices $x_u \in N_S(u)$ and $x_v \in N_S(v)$ with $u, v \in V_\alpha$ and
    $bc_S(x_u) \neq bc_S(x_v)$ then
        Let $X$ be a path in $G_\alpha$ between two such vertices $u$ and $v$ such that
        $X \setminus \{u, v\} \subseteq N_0$;
        Apply branching rule 3 to $X$; terminate;
    if $G[S \cup V_\alpha]$ contains a $K_4$-subdivision then
        Let $X \subseteq V_\alpha$ be a connected set such that $G[S + X]$ contains a
        $K_4$-subdivision;
        Apply branching rule 1 to $X$; terminate;
    if $\alpha$ is a P-node and $|V_\alpha^B| > \gamma(9)$ then
        $X = V_\alpha^B$ is a 4-protrusion (see Lemma 3);
        Apply the generic protrusion rule (reduction rule 1) with $X$; terminate;
    Mark the node $\alpha$;
```

We are now ready to prove:

Theorem 1. *The* K_4-minor cover *problem can be solved in* $2^{O(k)} \cdot n^{O(1)}$ *time.*

Proof. As observed at the beginning of this section, it is sufficient to show that one can solve disjoint K_4-minor cover in time $2^{O(k)} \cdot n^{O(1)}$. Let (G, S, k) be an input instance of disjoint K_4-minor cover. The recursive application of operations (a)-(e) (see the beginning of this subsection) to (G, S, k) produces a search tree Υ. From the fact that the various reduction and branching rules are safe, it follows that (G, S, k) is a YES-instance if and only if at least one of the leaf nodes in Υ corresponds to a YES-instance.

We now look at the time required to apply the operations (a)-(e) at each node of Υ. Every instance corresponding to a leaf node either is a trivial instance or is an independent instance (Theorem 2) which can be solved in $2^k \cdot n^{O(1)}$ time by applying branching rule 1 on pairs of vertices of F (Theorem 3). Clearly, the operations (a)–(c) can be applied in polynomial time. Consider the operation (d). The while-loop in the algorithm Branch-or-reduce iterates $O(n)$ times. At each iteration, we are in one of three situations: (i) we detect in polynomial time (Lemma 5) a connected subset X on which to apply one of the branching rules, or (ii) apply the protrusion rule in polynomial time (reduction rule 1), or (iii) none of these two cases occurs, and the node under consideration is marked.

Observe that by Lemma 5, the branching degree of the search tree is at most $2c_1 + 1$. To bound the size of Υ, we need the following claim.

Claim. In any application of branching rules 1–3, the measure μ strictly decreases.

Proof of claim. The statement holds for branching rule 1 since k reduces by one and $G[S]$ is unchanged. Recall that branching rules 2 and 3 put a vertex in the potential solution or add a path $X \subseteq F$ to S. In the first case, μ strictly decreases because k decreases and $\#cc(G[S])$ and $\#bc(G[S])$ remain unchanged. Let us see why μ strictly decreases also when we add a path X to S.

If branching rule 2 is applied, the number of biconnected components may increase by at most $2c_1 + 1$. This happens if every edge on the path X together with the two edges connecting the two end vertices of X to S add to the biconnected components of $G[S \cup X]$. So the new value of μ is $\mu' = (2c_1 + 2)k + (2c_1 + 2)\#cc(G[S \cup X]) + \#bc(G[S \cup X]) \leq (2c_1 + 2)k + (2c_1 + 2)(\#cc(G[S]) - 1) + (\#bc(G[S]) + 2c_1 + 1) \leq \mu - 1$. It remains to observe that an application of Branching rule 3 strictly decreases the number of biconnected components, and it does not increase the number of connected components. It follows that $\mu' \leq \mu - 1$. □

By this Claim, at every root-leaf computation path in Υ we have at most $\mu = (2c_1 + 2)k + (2c_1 + 2)\#cc(G[S]) + \#bc(G[S]) \leq (4c_1 + 5)k$ nodes at which a branching rule is applied. Since we branch into at most $(2c_1 + 1)$ ways, the number of leaves of Υ is bounded by $(2c_1 + 1)^{(4c_1+5)k}$. Also note that any root-leaf computation path contains $O(n)$ nodes at which a reduction rule is applied: This is because each reduction rule strictly decreases the size of the instance (and does not affect $G[S]$). It follows that the running time is bounded by $((4c_1 + 5)k + O(n)) \cdot (2c_1 + 1)^{(4c_1+3)k} \cdot poly(n) = 2^{O(k)} \cdot n^{O(1)}$. □

4 Conclusion and Open Problems

Due to the use of the generic protrusion rule (on t-protrusions for $t = 3$ or 4), the result in this paper is existential. A tedious case by case analysis would eventually lead to an explicit $c^k \cdot n^{O(1)}$ exponential FPT algorithm for some constant value c. It is an intriguing challenge to reduce the basis to a small c and/or get a simple proof of such an explicit algorithm. More generally, it would be interesting to investigate the systematic instantiation of protrusion rules.

We strongly believe that our method will apply to similar problems. The first concrete example is parameterized Outerplanar Vertex Deletion, or equivalently the $\{K_{2,3}, K_4\}$-minor cover problem. For that problem, we need to adapt the reduction and branching rules in order to preserve (respectively, eliminate) the existence of a $K_{2,3}$ as well. For example, the by-passing rule (reduction rule 3) may destroy a $K_{2,3}$ unless we only bypass a degree-two vertex when it is adjacent to another degree-two vertex. Similarly in reduction rule 6, we cannot afford to replace the set X by an edge. It would be safe with respect to $\{K_{2,3}, K_4\}$-minor deletion if, instead, X is replaced by a length-two path or by two parallel paths of length two (depending on the structure of X). We conjecture that for Outerplanar Vertex Deletion our reduction and branching rules can be adapted to design a single exponential FPT algorithm.

A more challenging problem would be to get a single exponential FPT algorithm for the treewidth-t vertex deletion for any value of t. Up to now and to the best of our knowledge, the fastest algorithm runs in $2^{O(k \log k)} \cdot n^{O(1)}$ time [12].

Acknowledgements. We would like to thank Saket Saurabh for his insightful comments on an early draft and Stefan Szeider for pointing out the application of this problem to Bayesian Networks.

References

1. Bidyuk, B., Dechter, R.: Cutset Sampling for Bayesian Networks. Journal of Artificial Intelligence Research 28, 1–48 (2007)
2. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)
3. Robertson, N., Seymour, P.: Graph minors XX: Wagner's conjecture. Journal of Combinatorial Theory B 92(2), 325–357 (2004)
4. Courcelle, B.: The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. Information and Computation 85, 12–75 (1990)
5. Downey, R., Fellows, M.: Parameterized complexity. Springer (1999)
6. Niedermeier, R.: Invitation to fixed parameter algorithms. Oxford University Press (2006)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
8. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. Journal of Computer and System Sciences 72(8), 1386–1396 (2006)
9. Dehne, F.K.H.A., Fellows, M.R., Langston, M.A., Rosamond, F.A., Stevens, K.: An $\mathrm{O}(2^{\mathrm{O}(\mathrm{k})}\mathrm{n}^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem. Theory of Computing Systems 41(3), 479–492 (2007)
10. Cao, Y., Chen, J., Liu, Y.: On Feedback Vertex Set New Measure and New Structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 93–104. Springer, Heidelberg (2010)
11. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Proceedings of FOCS 2011, pp. 150–159 (2011)
12. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Nearly optimal FPT algorithms for Planar-$\mathcal{F}$-Deletion (2011) (unpublished manuscript)
13. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters 32(4), 299–301 (2004)
14. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for feedback vertex set problems. Journal of Computer and System Sciences 74(7), 1188–1198 (2008)
15. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) Kernelization. In: Proceedings of FOCS 2009, pp. 629–638 (2009)

16. Fomin, F., Lokshtanov, D., Saurabh, S., Thilikos, D.: Bidimensionality and kernels. In: Proceedings of SODA 2010, pp. 503–510 (2010)
17. Fomin, F.V., Lokshtanov, D., Misra, N., Philip, G., Saurabh, S.: Hitting forbidden minors: Approximation and kernelization. In: Proceedings of STACS 2011, pp. 189–200 (2011)
18. Diestel, R.: Graph theory. Springer (2010)
19. Valdes, J., Tarjan, R., Lawler, E.: The recognition of series-parallel graphs. SIAM Journal on Computing 11, 298–313 (1982)
20. Bodlaender, H.L., de Fluiter, B.: Parallel Algorithms for Series Parallel Graphs. In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 277–289. Springer, Heidelberg (1996)
21. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Journal of Algorithms, 1–16 (1998)
22. Kim, E.J., Paul, C., Philip, G.: A Single-Exponential FPT Algorithm for the K_4-Minor Cover Problem. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 119–130. Springer, Heidelberg (2012), http://arxiv.org/abs/1204.1417
23. Joret, G., Paul, C., Sau, I., Saurabh, S., Thomassé, S.: Hitting and Harvesting Pumpkins. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 394–407. Springer, Heidelberg (2011)
24. Heggernes, P., van't Hof, P., Lokshtanov, D., Paul, C.: Obtaining a Bipartite Graph by Contracting Few Edges. In: Proceedings of FSTTCS 2011. LIPIcs, vol. 13, pp. 217–228 (2011)

An $O(n^3 \log\log n/\log^2 n)$ Time Algorithm for All Pairs Shortest Paths

Yijie Han[1] and Tadao Takaoka[2]

[1] School of Computing and Engineering, University of Missouri at Kansas City,
Kansas City, MO 64110, USA
hanyij@umkc.edu

[2] Department of Computer Science and Software Engineering,
University of Canterbury, Christchurch, New Zealand
T.Takaoka@cosc.canterbury.ac.nz

Abstract. We present an $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. This algorithm improves on the best previous result of $O(n^3(\log\log n)^3/\log^2 n)$ time.

Keywords: Algorithms, all pairs shortest paths, graph algorithms, upper bounds.

1 Introduction

Given an input directed graph $G = (V, E)$, the all pairs shortest path problem (APSP) is to compute the shortest paths between all pairs of vertices of G assuming that edge costs are real values. The APSP problem is a fundamental problem in computer science and has received considerable attention. Early algorithms such as Floyd's algorithm ([2], pp. 211-212) computes all pairs shortest paths in $O(n^3)$ time, where n is the number of vertices of the graph. Improved results show that all pairs shortest paths can be computed in $O(mn + n^2 \log n)$ time [9], where m is the number of edges of the graph. Pettie showed [14] an algorithm with time complexity $O(mn + n^2 \log\log n)$. See [15] for recent development. There are also results for all pairs shortest paths for graphs with integer weights[10,16,17,21,22,23]. Fredman gave the first subcubic algorithm [8] for all pairs shortest paths. His algorithm runs in $O(n^3(\log\log n/\log n)^{1/3})$ time. Fredman's algorithm can also run in $O(n^{2.5})$ time nonuniformly. Later Takaoka improved the upper bound for all pairs shortest paths to $O(n^3(\log\log n/\log n)^{1/2})$ [19]. Dobosiewicz [7] gave an upper bound of $O(n^3/(\log n)^{1/2})$ with extended operations such as normalization capability of floating point numbers in $O(1)$ time. Earlier Han obtained an algorithm with time complexity $O(n^3(\log\log n/\log n)^{5/7})$ [12]. Later Takaoka obtained an algorithm with time $O(n^3 \log\log n/\log n)$ [20] and Zwick gave an algorithm with time $O(n^3\sqrt{\log\log n}/\log n)$ [24]. Chan gave an algorithm with time complexity of $O(n^3/\log n)$ [6]. Chan's algorithm does not use tabulation and bit-wise parallelism. His algorithm also runs on a pointer machine.

What subsequently happening was very interesting. Takaoka thought that $O(n^3/\log n)$ could be the ultimate goal and raised the question [20] whether

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 131–141, 2012.

$O(n^3/\log n)$ can be achieved. Chan first achieved $O(n^3/\log n)$ time and also thought that $O(n^3/\log n)$ is a natural bound [6]. However, Han showed an algorithm with $O(n^3(\log\log n/\log n)^{5/4})$ time [11]. Because in [11] Han exhausted Takaoka's technique [19] Han thought that this result should be it and did not expect any improvements in the near future (see the reasoning Han gave in the paper [11] explaining why it should be difficult to further improve). However, Chan came up with an algorithm with time complexity $O(n^3(\log\log n)^3/\log^2 n)$ [5]. Chan [5] believes that $O(n^3/\log^2 n)$ is probably the final chapter. Our experience indicates that Chan may be correct. Here we present an algorithm with time complexity $O(n^3\log\log n/\log^2 n)$. Thus we further remove a factor of $(\log\log n)^2$ from the time complexity of the best previous result due to Chan.

We would like to point out the previous results which influenced the formation of our ideas presented in this paper. They are: Floyd's algorithm [2], Fredman's algorithm [8], Takaoka's algorithm [19], Han's algorithm [11], Chan's algorithm [5].

2 Preparation

Since it is well known that the all pairs shortest paths computation has the same time as computing the distance product of two matrices [1] ($C = AB$), we will concentrate on the computation of distance product.

We divide the first $n \times n$ matrix A into t_1 submatrices $A_1, A_2, , A_{t_1}$ each having dimension $n \times n/t_1$, where $t_1 = n^{1-r_1}$ and r_1 is a constant to be determined later. We divide the second $n \times n$ matrix B into t_1 submatrices $B_1, B_2, ..., B_{t_1}$ each having dimension $n/t_1 \times n$. Therefore $C = AB = A_1B_1 + A_2B_2 + ... + A_{t_1}B_{t_1}$, where $*$ is addition and $+$ is the minimum operation. In the following we consider the computation of the distance product of an $n \times n/t_1$ matrix E with a $n/t_1 \times n$ matrix F. The reason we need to do the division for this level will be understood later in this paper.

We then divide the $n \times n/t_1$ matrix E into $t_2 = (n/t_1)/(r_2\log n/\log\log n)$ submatrices $E_1, E_2, ..., E_{t_2}$ each having dimension $n \times (r_2\log n/\log\log n)$, where r_2 is a constant to be determined later. Similarly we divide the $n/t_1 \times n$ matrix F into t_2 submatrices each having dimension $(r_2\log n/\log\log n) \times n$. Now $EF = E_1F_1 + E_2F_2 + ... + E_{t_2}F_{t_2}$.

In the following we will first consider the computation of E_1F_1, and then the computation of EF (or A_1B_1). After that it takes $O(n^2t_1)$ time straightforwardly to get the all-pairs shortest path of the input graph.

Let $E_1 = (e_{ij})$ and $F_1 = (f_{ij})$. We will first, for each $1 \leq i, j \leq r_2\log n/\log\log n$, sort $f_{ik} - f_{jk}$, $k = 1, 2, ..., n$. After sorting each $f_{ik} - f_{jk}$ has a rank in $[1, n]$. We then give $f_{ik} - f_{jk}$ a label $l_f(i, j, k) = l$ if $f_{ik} - f_{jk}$ has rank in $(l \cdot n/\log^9 n, (l+1) \cdot n/\log^9 n]$. $l_f(i, j, k)$ uses $9\log\log n$ bits. For each $e_{kj} - e_{ki}$ we will give it label $l_e(k, i, j) = l$ if $f_{ik_1} - f_{jk_1} < e_{kj} - e_{ki} \leq f_{ik_2} - f_{jk_2}$, where $f_{ik_1} - f_{jk_1}$ has rank (not label) $l \cdot n/\log^9 n$ and $f_{ik_2} - f_{jk_2}$ has rank $(l+1) \cdot n/\log^9 n$. $l_e(k, i, j)$ also uses $9\log\log n$ bits.

According to Fredman [8] and Takaoka [19], if the labels of $e_{k_1j} - e_{k_1i}$ and $f_{ik_2} - f_{jk_2}$ are different then we can determine either $e_{k_1i} + f_{ik_2} < e_{k_1j} + f_{jk_2}$ or $e_{k_1i} +$

$f_{ik_2} > e_{k_1j}+f_{jk_2}$. Say $e_{k_1j}-e_{k_1i}$ has label l_e and $f_{ik_2}-f_{jk_2}$ has label l_f. If $l_e < l_f$ ($l_f < l_e$) then $e_{k_1j} - e_{k_1i} < f_{ik_2} - f_{jk_2}$ ($e_{k_1j} - e_{k_1i} > f_{ik_2} - f_{jk_2}$) and therefore $e_{k_1j}+f_{jk_2} < e_{k_1i}+f_{ik_2}$ ($e_{k_1j}+f_{jk_2} > e_{k_1i}+f_{ik_2}$). However, when their labels are the same then we cannot determine this. However, only a fraction $1/\log^9 n$ of the total number of $(e_{k_1j} - e_{k_1i})$'s (one out of all lables) are undetermined for each $f_{ik_2} - f_{jk_2}$ and therefore overall a fraction of $1/\log^9 n$ of all pairs of $e_{k_1j} - e_{k_1i}$ and $f_{ik_2} - f_{jk_2}$ are undetermined. In case of indeterminacy for two indices i, j we will pick i over j (to include i in the computation) when $i < j$ and leave the j-th position (or index) to be computed separately. This separated computation can be done in brute force and it takes $O(n^3/\log^8 n)$ time for the whole computation, i.e. the computation of AB. The actual counting of complexity of this separate computation is as this: There are $w = O(n^3 \log n/\log\log n)$ pairs of $e_{k_1j} - e_{k_1i}$ and $f_{jk_2} - f_{ik_2}$ (k_1 and k_2 each take value in $[0..n-1]$ and thus have a factor of $x = n^2$, there are $y = n \log\log n/(r_2 \log n)$ pairs of E and F for each pair of k_1 and k_2 and for each pair of E and F there are $z = O((r_2 \log n/\log\log n)^2)$ pairs of $e_{k_1j} - e_{k_1i}$ and $f_{jk_2} - f_{ik_2}$. $w = xyz$. Because $1/\log^9 n$ of these pairs are in the separate computation the complexity of the separate computation takes $O((n^3 \log n/\log\log n) \cdot (1/\log^9 n)) = O(n^3/\log^8 n)$ time.

3 The Further Details

Now for fixed i and k we pack $l_e(k, i, j)$, $j = 1, 2, ..., r_2 \log n/\log\log n$, into one word and call it $l_e(k, i)$. This can be done when r_2 is small. Also for fixed i and k we pack $l_f(i, j, k)$, $j = 1, 2, ..., r_2 \log n/\log\log n$, into one word and call it $l_f(i, k)$. By comparing $l_e(k_1, i)$ and $l_f(i, k_2)$ we can let the result be 1 if index i should be chosen over all the other $r_2 \log n/\log\log n - 1$ indices, and let it be 0 otherwise. This computation of comparing one index with all the other $r_2 \log n/\log\log n$ indices is done in constant time by concatenating $l_e(k_1, i)$ and $l_f(i, k_2)$ into one word of less than $\log n$ bits and then index into a precomputed table to get the result of either 0 or 1.

Note that since $l_e(k, i)$ has $9r_2 \log n$ bits, and we will pick r_2 later such that $9r_2 \log n$ is much smaller than $\log n$ and therefore $l_e(k, i), k = 1, 2, ..., n$ can be sorted into $t_3 = 2^{9r_2 \log n} = n^{9r_2}$ blocks such that each block has the same word $(l_e(k, i))$ value.

For the purpose of computing E_1F_1, there are $r_2 \log n/\log\log n$ i's (columns in E_1) and for each (column) of them we have n $l_e(k, i)$'s (one on each row) and these $l_e(k, i)$'s ($0 \le k < n$) form t_3 blocks and for each of these blocks we get a $1 \times n$ vector of bits (0's and 1's) that are the result of the above computation (i.e. indexing into a table to get a 0 or a 1). We need to compute these (a vector of n) 0's and 1's for one $l_e(k, i)$ in a block because all $l_e(k, i)$'s in a block have the same value. Thus we need to compute $r_2(\log n/\log\log n)t_3$ vectors (of n bits each), and this can be done in $O(r_2(\log n/\log\log n)t_3 n)$ time (one step gets one bit by the above table lookup computation).

On the average for each such an n bit vector v there are only $n/(r_2 \log n/\log \log n)$ bits that are 1's and the remaining bits are 0's. This is so because of the way we formed $l_e(k_1, i)$ and $l_f(i, k_2)$ and the way that we stipulate that the result of comparison of $l_e(k_1, i)$ and $l_f(i, k_2)$ is 0 or 1. We take v and first divide it into $n/((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))$ small vectors each with dimension $1 \times ((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))$, where r_3 is a constant to be determined later. Now, on the average, each small vector has $r_3 \log n/\log\log n$ bits which are 1's. If a small vector v' has between $(t-1)r_3 \log n/\log\log n + 1$ and $tr_3 \log n/\log\log n$ bits of 1's we will make a set V of t small vectors each having $((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))$ bits and containing a subset of no more than $r_3 \log n/\log\log n$ 1's from v'.

Because the multiplication of each row of E_1 with all columns of F_1 results in $r_2 \log n/\log\log n$ vectors having a total of n bits of 1's, they will result in $2n(r_2 \log n/\log\log n)/((r_2 \log n/\log\log n)(r_3 \log n/\log$ $\log n))= 2n/(r_3 \log n/\log\log n)$ small vectors, where factor 2 is because of some small vectors may have less than $r_3 \log n/\log\log n$ bits of 1's.

For fixed i (a row of F_1) (and therefore $l_f(i, k)$, $0 \leq k < n$) and fixed value of $l_e(k, i)$'s (a block) we formed $2n/((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))$ small vectors each having $(r_2 \log n/\log\log n)(r_3 \log n/\log\log n)$ bits with no more than $r_3 \log n/\log\log n$ bits are 1's. Therefore each small vector can be represented by a word (with no more than $\log n$ bits) when r_3 is small. This is so because $\sum_{t=0}^{r_3 \log n/\log\log n} \binom{(r_2 \log n/\log\log n)(r_3 \log n/\log\log n)}{t} < n$. We first form these $2n/((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))$ words for each vector (on the average) and then duplicate these words for all rows in the block because all rows in the same block has the same $l_e(k, i)$ value. The reason we do this duplicating is to save time because small vectors with the same value need not to be computed into words repeatedly. Thus for the multiplication of E_1F_1 we obtained $2n^2/(r_3 \log n/\log\log n)$ words. And for the multiplication of A_1B_1 we obtained $2n^{2+r_1}/((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))$ words. And therefore for the multiplication of AB we have obtained $O(n^3(\log\log n/\log n)^2)$ words and computation thus far takes $O(n^3(\log\log n/\log n)^2)$ time.

However these $O(n^3(\log\log n/\log n)^2)$ words contain more than $O(n^3(\log\log n/\log n)^2)$ indices because multiple indices are coded into one word. Thus we shall combined these words to cut the number of indices.

To further combine these words we need to consider only the words formed by $E_i[1, 1..(r_2 \log n/\log\log n)] \times F_i[1..(r_2 \log n/\log\log n),$ $1..(r_2 \log n/\log\log n)(r_3 \log n/\log\log n)]$ (there are $r_2 \log n/\log\log n$ resulting words out of this multiplication as we explained above), $i = 1, 2, ..., n^{r_1}/(r_2 \log n/\log\log n)$. Thus there are n^{r_1} words. We need to reduce them to $n^{r_1}/(r_3 \log n/\log\log n)$ words in $O(n^{r_1})$ time and thereafter we can simply disassemble indices (for minimum) out of packed words and finish the remaining computation straightforwardly. Because each word w contains a set S_w of no more than $r_3 \log n/\log\log n$ columns (these columns have 1's and the other columns have 0's) in F_i there are $\sum_{l=1}^{r_3 \log n/\log\log n} \binom{((r_2 \log n/\log\log n)(r_3 \log n/\log\log n))}{l}$ $\leq n^{cr_3}$ choices, where c is a constant. When r_3 is much smaller than r_1 there are many words among the n^{r_1} words having the same S_w sets. This is the fact we can take advantage of. In the following we will refer two of these words with the same

S_w sets as w_1 and w_2, i.e., the two small vectors represented by w_1 and w_2 are the same (equal or identical).

4 Combining Words

The scheme for combining words is a little complicated. The basic idea is that, since we have indices gathered in $O(n^3(\log\log n/\log n)^2)$ words, we just need to do pair-wise comparisons between pairs of indices (paths) to reduce the number of indices by half. If we do this for $2\log\log n$ rounds we can reduce the number of indices to $n^3/(\log n)^2$ and then we can just disassemble indices from words and finish the computation straightforwardly in $O(n^3/(\log n)^2)$ time. Note that because we do pairing-off the time complexity will remain to be $O(n^3(\log\log n/\log n)^2)$.

The complication of our algorithm comes from the fact that indices are encoded in $O(n^3(\log\log n/\log n)^2)$ words. To deal with this encoding we have to design an algorithm that utilizes the special characteristics of the encoding.

We use a different labeling for the matrix $B_1 = (b_{ij})$ and $A_1 = (a_{ij})$. We will, for each $1 \le i, j \le n^{r_1}$, sort $b_{ik} - b_{jk}$ together with $a_{kj} - a_{ki}$, $k = 1, 2, ..., n$. For A and B the total time for sorting is $O(n^{2+r_1}\log n)$. This gives the rank of $b_{ik} - b_{jk}$ $(a_{kj} - a_{ki})$ which we denote by $l_{b_1}(i,j,k)$ $(l_{a_1}(k,i,j))$. These ranks take value from 1 to $2n$ and have $\log n + 1$ bits. There are n^{2r_1} choices of i, j pairs and for each of these choices (each pair of i and j) and for each set

$U_t = \{t(r_2\log n/\log\log n)(r_3\log n/\log\log n) + 1,$
$t(r_2\log n/\log\log n)(r_3\log n/\log\log n) + 2, ...,$
$(t+1)(r_2\log n/\log\log n)(r_3\log n/\log\log n)\},$

$t = 1, 2, ..., n/((r_2\log n/\log\log n)(r_3\log n/\log\log n))$, where values of k are taken, choose any subset of U_t containing no more than $r_3\log n/\log\log n$ elements and there are no more than $(n/((r_2\log n/\log\log n)(r_3\log n/\log\log n))$ $* \sum_{l=1}^{r_3\log n/\log\log n}\binom{((r_2\log n/\log\log n)(r_3\log n/\log\log n))}{l} \le n^{1+cr_3}$ choices, where c is a constant, and thus there are a total of $n^{1+2r_1+cr_3}$ choices for all pairs of i and j. For each of these choices we use a word w to represent the total of (a) a choice of the pair i, j, (b) a choice (referred to as d) of the subset of U_t ($n^{2r_1+cr_3}$ choices) and (c) the packing of no more than $r_3\log n/\log\log n$ ranks ($l_{b_1}(i,j,k)$'s) (where k take values over elements in the subset of U_t). Note that straightforward packing will not work because it will take $O(\log^2 n/\log\log n)$ bits (a subset of U_t has up to $O(\log n/\log\log n)$ elements and each element corresponds to $O(\log n)$ bits of an $l_{b_1}(i,j,k)$.) and cannot be stored in a word of $\log n$ bits. What we do is first to build a trie for the $r_3\log n/\log\log n$ ranks. An example of such a trie is shown in Fig. 1(a). This trie is a binary tree with a node having two children when there are ranks with the most significant bits differ at the node's level. Next we build a packed trie by removing nodes v with only one child except the root. The edge connecting this removed node and its child is also removed. This is shown in Fig. 1(b). Thus let $v_1, v_2, ..., v_t$ be such a chain with v_i being

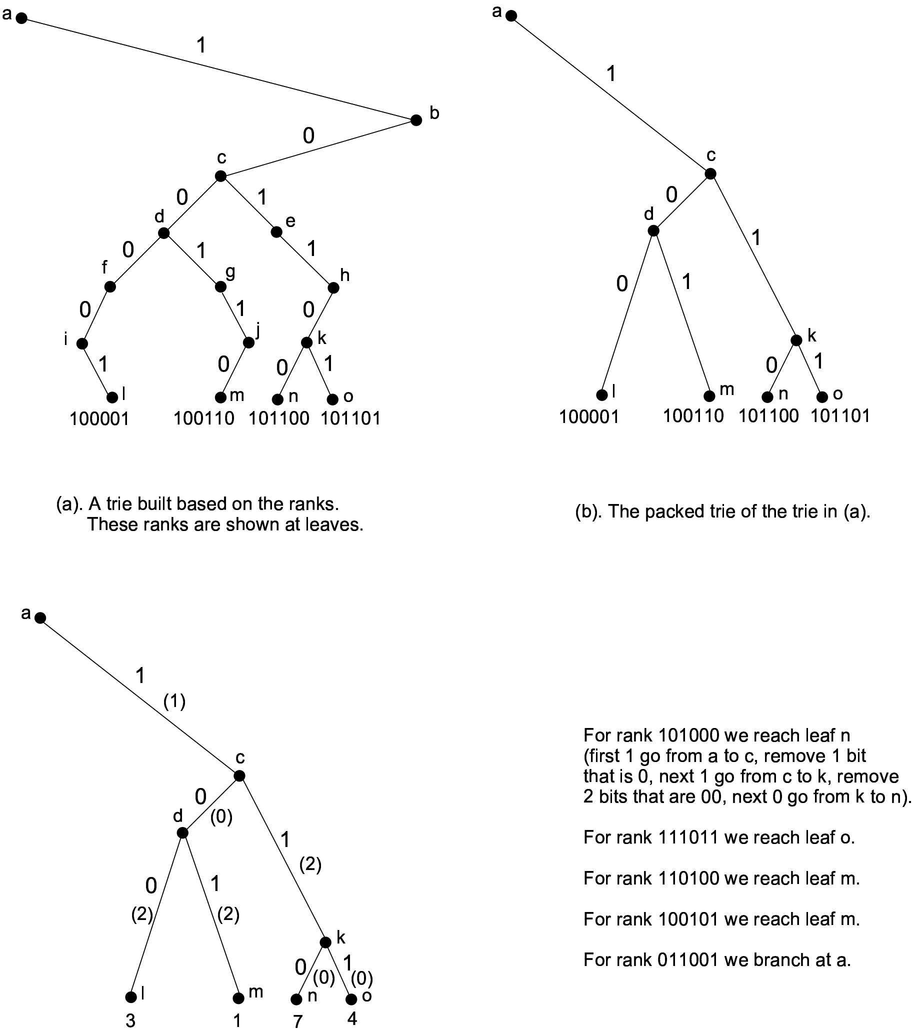
v_{i+1}'s parent, v_1 and v_t having two children and v_i, $i \neq 1, t$, having one child, and we will remove $v_2, v_3, ..., v_{t-1}$. Edges $(v_2, v_3), (v_3, v_4), ..., (v_{t-1}, v_t)$ are also removed. The edge originally connecting v_1 and v_2 are now made to connect v_1 and v_t. We will record on edge (v_1, v_t) that $t-2$ edges (bits) are removed. Also at leaves, we store only relative address of k (having value between 1 and $(r_2 \log n / \log\log n)(r_3 \log n / \log\log n))$ instead of the value of $l_{b_1}(i, j, k)$ (having value between 1 and $2n$). Such a packed trie is shown in Fig. 1(c) and it can be stored in a word w with $c \log n$ bits, where c is a constant less than 1.

Now with $l_{a_1}(1, i, j)$, we can follow this packed trie in word w and reach a leaf l of the trie (by starting at the root, repeatedly deleting corresponding bits which has been removed from the packed trie and following the path in the packed trie). In Fig. 1(d) we show such situations. Therefore we can precompute a table T_1 and use the values of $l_{a_1}(1, i, j)$ and w to index into T_1 to get leaf l. (Note that $l_{a_1}(1, i, j)$ has $\log n + 1$ bits but this can be reduced to $c \log n$ bits with $c < 1$ by using multiple tables replacing T_1 and taking care of a fraction of $\log n + 1$ bits at a time). Here we will use l to represent both the leaf and the relative address of k mentioned in immediate previous paragraph. From l we get the value of k and we can then compare $l_{a_1}(1, i, j)$ and $l_{b_1}(i, j, k)$ to find the most significant bit where $l_{a_1}(1, i, j)$ and $l_{b_1}(i, j, k)$ differ (this can be done by exclusive-oring the two values and then finding the most significant bit which is 1 by indexing into a precomputed table). Say this bit is the b-th most significant bit. By using the values of b, $l_{a_1}(1, i, j)$ and w (indexing into another precomputed table T_2) we can then figure out at which chain C of the original trie $l_{a_1}(1, i, j)$ "branches out". Note that we need NOT to know the value of C. We need know only within C the branch happens and whether it branches to left or to right (these information can be figured out with b, $l_{a_1}(1, i, j)$ and w.) Thus the output of indexing into T_2 can be the columns where index i should be taken over index j.

We can store w as an array element in an array W as $W[i][j][t][d]$, here, i, j, t, d are the ones mentioned in the first paragraph of this section. We need not to pack $l_{a_1}(k, i, j)$'s because only one $l_{a_1}(k, i, j)$ is considered at a time.

Now for the above mentioned words w_1 and w_2 (last paragraph of Section 3) we can get t and d value (both w_1 and w_2 are associated with the same t value because we put them together as explained above, w_1 and w_2 are associated with the same d value because as we mentioned above that we can get this by setting r_3 much smaller than r_1). We can also get i from w_1 and j from w_2. Thus we can get $W[i][j][t][d]$. Now we use the values of $W[i][j][t][d]$ and $l_{a_1}(1, i, j)$ to index into precomputed table T_1, T_2 to get rid of half of indices in w_1 plus w_2. Repeating this process we can then remove a factor of $r_3 \log n / \log\log n$ from the total number of indices. Thereafter we disassemble the indices from packed such that one word contains one index and the remaining computation can be carried out easily.

The precomputation of tables T_1, T_2 is not difficult and its complexity cannot dominate the overall computation. The reason is because all these computations



(a). A trie built based on the ranks.
These ranks are shown at leaves.

(b). The packed trie of the trie in (a).

(c). Packed trie. Numbers in parenthesis are the number of edges (bits) removed from the chain. The number at the leaves are relative address of columns of the matrix.

For rank 101000 we reach leaf n (first 1 go from a to c, remove 1 bit that is 0, next 1 go from c to k, remove 2 bits that are 00, next 0 go from k to n).

For rank 111011 we reach leaf o.

For rank 110100 we reach leaf m.

For rank 100101 we reach leaf m.

For rank 011001 we branch at a.

(d).

Fig. 1.

have polynomial complexity and we can reduce the table size to n^ϵ with ϵ being an arbitrarily small constant.

The choice of r_1, r_2, r_3 should not be difficult now. The complexity of the algorithm as we described above is $O(n^3(\log\log n)^2/\log^2 n)$.

5 Removing Another Factor of $\log\log n$ from Complexity

In our algorithm we partitioned the number of rows of E_1 and the number of columns of F_1 by a factor of $O(\log^c n)$ at a time. This gives $O(\log n/\log\log n)$ depths of partitioning and results in the loss of a factor of $\log\log n$ in time complexity. If we partition the number of rows of E_1 and the number of columns of F_1 by a constant factor at a time our algorithm would have $O(\log n)$ depths of partitioning and thus can remove another factor of $\log\log n$ from time complexity. If we do this way the rows of E_1 and columns of F_1 are not partitioned uniformly. Such modification does not involve new ideas and does not require drastically different time analysis. The principle and the approach remain intact, only the parameter of the algorithm changed. The details of this modification is as follows.

Let $E_1 = (e_{ij})$ and $F_1 = (f_{ij})$. We will first, for each $1 \leq i, j \leq c_1 \log n$ ($c_1 < 1$ is a constant), sort $f_{ik} - f_{jk}$, $k = 1, 2, ..., n$. After sorting each $f_{ik} - f_{jk}$ has a rank in $[1, n]$. We then give $f_{ik} - f_{jk}$ a label $l_f(i, j, k) = l$ if $f_{ik} - f_{jk}$ has rank in $(l \cdot n/2, (l + 1) \cdot n/2]$. $l_f(i, j, k)$ uses 1 bit. For each $e_{kj} - e_{ki}$ we will give it label $l_e(k, i, j) = l$ if $f_{ik_1} - f_{jk_1} < e_{kj} - e_{ki} \leq f_{ik_2} - f_{jk_2}$, where $f_{ik_1} - f_{jk_1}$ has rank (not label) $l \cdot n/2$ and $f_{ik_2} - f_{jk_2}$ has rank $(l + 1) \cdot n/2$.

Now if $l_e(k_1, i, j) \neq l_f(i, j, k_2)$ then we can decide to discard an index. If $l_e(k_1, i, j) = l_f(i, j, k_2)$ then we cannot make a decision. If we group elements of the same label together then the above labeling partitions the array $[0..n-1, 0..n-1]$ into 4 divisions and among them there are 2 divisions we can determine the index for the shorter path and discard the other index and for the other 2 divisions we cannot determine. The area for the determined divisions is $n^2/2$ and the area for the undetermined divisions is also $n^2/2$. Now for the undetermined divisions we sort and label elements again and further partitions. In this way when we partitioned to $c \log\log n$ levels for a constant c then the area of undetermined divisions is $n^2/\log^c n$. See Fig. 2 (Fig. 2 may looks like a Quad tree and actually it is not).

Built on top of the above partition we now do $l_e(k_1, i, j + 1)$ and $l_f(i, j + 1, k_2)$. This will further partition the divisions. However, once the undetermined divisions area reaches $n^2/\log^c n$ we will stop and not further partition it. Thus the undetermined divisions obtained when we worked on $l_e(k_1, i, j)$ and $l_f(i, j, k_2)$ are not further partitioned.

We can continue to work on $l_e(k_1, i, j + 2)$ and $l_f(i, j + 2, k_2)$, $l_e(k_1, i, j + 3)$ and $l_f(i, j + 3, k_2)$, ..., $l_e(k_1, i, j + c_1 \log n)$ and $l_f(i, j + c_1 \log n, k_2)$. Note the difference between here and the algorithm we gave before in the paper. Before

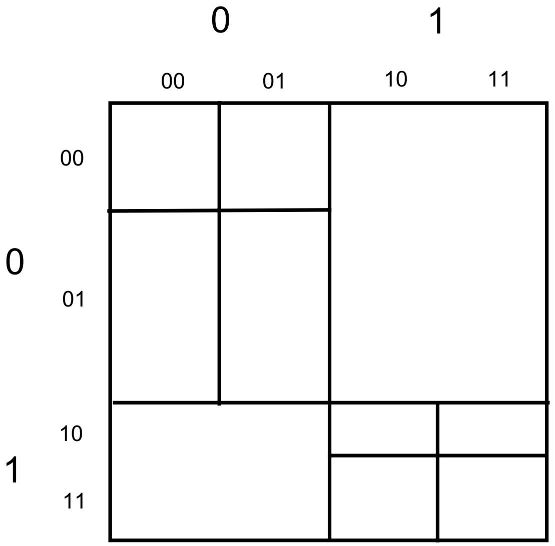


Fig. 2.

we can only go to $l_e(k_1, i, j+r_2 \log n/\log\log n)$ and $l_f(i, j+r_2 \log n/\log\log n, k_2)$ (i.e. combining about $\log n/\log\log n$ columns (rows) of E_1 (F_1)), now we can go to $l_e(k_1, i, j+c_1 \log n)$ and $l_f(i, j+c_1 \log n, k_2)$ (i.e. combining about $\log n$ columns (rows) of E_1 (F_1)). This is the key for us to remove a factor of $\log\log n$ in time complexity.

In each partitioned division the winning index for the shortest paths is the same. The remaining computation basically follows the algorithm given before in the paper. First, for each row in E_1 and consecutive $r_3 \log n/\log\log n$ columns (consecutive in matrix F_1) we use a word w_1 to indicate the $r_3 \log n/\log\log n$ winning indices. Now compare words w_i (obtained for the same row in E_i and the same columns in F_i). If w_i and w_j are equal we then combine them into one word (removing half of the indices) by table lookup. We keep doing this until we combined $\log n/\log\log n$ words into one word. Thus now each word has $r_3 \log n/\log\log n$ winning indices and they are combined from $\log^3 n/(\log\log n)^2$ indices. Thus thereafter we can disassemble the indices from the word and the remaining computation shall take $O(n^3 \log\log n/\log^2 n)$ time.

Theorem: All pairs shortest paths can be computed in $O(n^3 \log\log n/\log^2 n)$ time.

As pointed by Chan that $O(n^3/\log^2 n)$ may be the final chapter and we are $\log\log n$ factor shy of this. We will leave this to future research. Also a referee pointed out that if our scheme can be recursively exploited like in Strassen's algorithm [18] then $O(n^{2+\epsilon})$ time may be achieved.

Acknowledgment. The first author would like to thank Professor Timothy M. Chan for sending him the paper [5] upon his request. We are also thankful to referees' helpful comments on the paper.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley, Reading (1983)
3. Albers, S., Hagerup, T.: Improved parallel integer sorting without concurrent writing. Information and Computation 136, 25–51 (1997)
4. Batcher, K.E.: Sorting networks and their applications. In: Proc. 1968 AFIPS Spring Joint Summer Computer Conference, pp. 307–314 (1968)
5. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proc. 2007 ACM Symp. Theory of Computing, pp. 590–598 (2007)
6. Chan, T.M.: All-Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 318–324. Springer, Heidelberg (2005)
7. Dobosiewicz, W.: A more efficient algorithm for min-plus multiplication. Inter. J. Comput. Math. 32, 49–60 (1990)
8. Fredman, M.L.: New bounds on the complexity of the shortest path problem. SIAM J. Computing 5, 83–89 (1976)
9. Fredman, M.L., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34, 596–615 (1987)
10. Galil, Z., Margalit, O.: All pairs shortest distances for graphs with small integer length edges. Information and Computation 134, 103–139 (1997)
11. Han, Y.: An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest paths. Algorithmica 51(4), 428–434 (2008)
12. Han, Y.: Improved algorithms for all pairs shortest paths. Information Processing Letters 91, 245–250 (2004)
13. Han, Y.: A note of an $O(n^3/\log n)$ time algorithm for all pairs shortest paths. Information Processing Letters 105, 114–116 (2008)
14. Pettie, S.: A Faster All-Pairs Shortest Path Algorithm for Real-Weighted Sparse Graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 85–97. Springer, Heidelberg (2002)
15. Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM J. Comput. 34(6), 1398–1431 (2005)
16. Sankowski, P.: Shortest Paths in Matrix Multiplication Time. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 770–778. Springer, Heidelberg (2005)
17. Seidel, R.: On the all-pairs-shortest-path problem in unweighted undirected graphs. J. Comput. Syst. Sci. 51, 400–403 (1995)
18. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik 14(3), 354–356 (1969)
19. Takaoka, T.: A new upper bound on the complexity of the all pairs shortest path problem. Information Processing Letters 43, 195–199 (1992)

20. Takaoka, T.: An $O(n^3 \log\log n/\log n)$ time algorithm for the all-pairs shortest path problem. Information Processing Letters 96, 155–161 (2005)
21. Thorup, M.: Undirected single source shortest paths with positive integer weights in linear time. Journal of ACM 46(3), 362–394 (1999)
22. Yuster, R., Zwick, U.: Answering distance queries in directed graphs using fast matrix multiplication. In: 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 389–396. IEEE Comput. Soc., Los Alamitos (2005)
23. Zwick, U.: All pairs shortest paths using bridging sets and rectangular matrix multiplication. Journal of the ACM 49(3), 289–317 (2002)
24. Zwick, U.: A Slightly Improved Sub-cubic Algorithm for the All Pairs Shortest Paths Problem with Real Edge Lengths. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 921–932. Springer, Heidelberg (2004)

Linear Time Algorithm for Computing a Small Biclique in Graphs without Long Induced Paths

Aistis Atminas[1], Vadim V. Lozin[2], and Igor Razgon[3]

[1] DIMAP and Mathematics Institute, University of Warwick, Coventry CV4 7AL, UK
A.Atminas@warwick.ac.uk

[2] DIMAP and Mathematics Institute, University of Warwick, Coventry CV4 7AL, UK
V.Lozin@warwick.ac.uk

[3] Department of Computer Science, University of Leicester, University Road, Leicester, LE1 7RH, UK
ir45@mcs.le.ac.uk

Abstract. The biclique problem asks, given a graph G and a parameter k, whether G has a complete bipartite subgraph of k vertices in each part (a biclique of order k). Fixed-parameter tractability of this problem is a longstanding open question in parameterized complexity that received a lot of attention from the community. In this paper we consider a restricted version of this problem by introducing an additional parameter s and assuming that G does not have induced (i.e. chordless) paths of length s. We prove that under this parameterization the problem becomes fixed-parameter linear. The main tool in our proof is a Ramsey-type theorem stating that a graph with a long (not necessarily induced) path contains either a long *induced* path or a large biclique.

1 Introduction

Overview of Our Results. Let us call a complete bipartite graph $H = (A, B, E)$ with $|A| = |B| = k$ a *biclique* of order k. Given a graph G and parameter k, the Biclique problem asks if G has a biclique of order k as a subgraph. Fixed-parameter tractability of this problem is a longstanding open question that received significant attention from the parameterized complexity community and is believed to be W[1]-hard (see the abstract of [6]).

In this paper we consider a restricted version of this problem by introducing an additional parameter s and assuming G to be P_s-free, i.e. without induced paths of length s. We show that under this additional parameterization the biclique problem becomes fixed-parameter linear. Let us remark that the parameterization by s alone is not enough for efficient computing of a largest biclique (Proposition 1). Indeed, the construction used by Johnson [16] to establish the NP-hardness of the Biclique problem in fact reduces the Clique problem to an instance of the Biclique problem on P_8-free graphs, that is the Biclique problem is NP-hard on P_s-free graphs for $s \geq 8$. In this sense, the use of s as an additional parameter is meaningful.

The key ingredient in our solution is a combinatorial statement (Theorem 1) claiming the existence of a number $A(k, s)$ such that every P_s-free graph with a path of length at least $A(k, s)$ has a biclique of order k. This result belongs to a large body of

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 142–152, 2012.

'Ramsey-type' theorems showing that if the given graph is 'large' in a certain sense, then it contains a large subgraph (either induced or not) that belongs to one of the specified families. In our case, the largeness condition is 'long path' and the families are bicliques and induced (i.e. chordless) paths. The proof of this result requires a number of intermediate stages which are proven by a non-trivial use of classical Ramsey's theorem. In particular, we use a 'non-binary' form of Ramsey's theorem with hyperedges of size 3, although we apply it to simple graphs only.

Related Work. The Biclique problem appears under the name of 'Balanced Complete Bipartite Subgraph' as problem [GT24] in the famous book of Garey and Johnson, an NP-hardness proof has been further provided by Johnson in [16]. An application of the problem to the VLSI design is described in detail in [1]. The problem has been considered in the context of approximation [12] and exact exponential time algorithms [3]. Polynomial time algorithms for a number of restricted classes of the Biclique problem have been proposed in [1]. To the best of our knowledge, the question regarding fixed-parameter tractability of Biclique was first asked in [9]. The question has been restated as an open problem in a number of subsequent publications, see e.g. [6], where the complexity of a number of parameterized problems is characterised as Biclique-hard. The *induced* Biclique problem is known to be W[1]-hard [7,13].

Graphs without long induced paths, i.e. P_s-free graphs for a constant s, have been extensively studied in the literature (see e.g. [2,11,22]). For small values of s, the structure of P_s-free graphs is simple. For instance, P_3-free graphs are precisely the graphs every connected component of which is a clique. P_4-free graphs also enjoy many nice properties. In particular, the clique-width of P_4-free graphs is bounded by a constant and hence many algorithmic problems that are generally NP-hard admit polynomial time solutions when restricted to P_4-free graphs.

In the class of P_s-free graphs with $s \geq 5$, the situation changes drastically and the computational complexity changes from polynomial-time solvability to NP-hardness for many important algorithmic graph problems. For instance, VERTEX COLORING [18] and MINIMUM DOMINATING SET [17] are NP-hard for P_5-free graphs, and VERTEX 4-COLOURABILITY is NP-hard for P_8-free graphs [5]. For many other problems, the complexity status on graphs without long induced paths is unknown. For instance, the complexity status is unknown for MAXIMUM INDEPENDENT SET in P_s-free graphs with $s \geq 5$ and for VERTEX 3-COLOURABILITY in P_s-free graphs with $s \geq 7$ (for some partial results related to these problems we refer the reader to [15,20,21,24,25]).

Structure of the Paper. Section 2 presents the algorithm for computing a biclique, Section 3 proves the main combinatorial result, Section 4 discusses directions of further research. All graphs in this paper are undirected, without loops and multiple edges.

2 Computing a Small Biclique in a Graph without Long Induced Paths

A biclique of order k is a bipartite graph (A, B, E) with $|A| = |B| = k$ and every $u \in A$ being adjacent to every $v \in B$. A notorious problem in Parameterized Complexity asks, given a parameter k, if the given graph has a biclique of order k. The fixed-parameter tractability of this problem is wide open despite efforts of many researchers. In this paper we consider the following restricted version of this problem (the abbreviation NLIP in the name of this problem stands for 'No Long Induced Paths').

NLIP-BICLIQUE
Input: A graph G
Parameters: k, s
Assumption: G is P_s-free
Output: A biclique of G of size at least k or 'NO' if there is no such biclique.

The following proposition, essentially proven in [16] shows that the choice of parameters is meaningful in the sense that s alone is not enough to compute a maximum biclique efficiently.

Proposition 1. *Computing maximum biclique in a P_s-free graph is NP-hard for $s \geq 8$*

Proof. This is implicitly proven in [16] because an instance of the Clique problem is reduced to an instance of the Biclique problem on a P_8-free graph. Indeed, by construction, given a graph G, a bipartite graph H is constructed in which the first part A corresponds to the edges of G and the second part B corresponds to a superset of its vertices and each vertex of A is adjacent to all vertices of B but those corresponding to the endpoints of the respective edge. It is not hard to see that H is P_8-free. Indeed, let P be a path of length 8. It has one terminal vertex u in A, and 4 vertices $v_1, \dots, v_4$ of B included in it. Three vertices out of $v_1, \dots, v_4$ are non-adjacent to u in P but only two of them may be the endpoints of the respective edge. It follows that u is necessarily adjacent in H to the remaining one, thus producing a chord in P. □

In this paper we prove that the NLIP-BICLIQUE problem is FPT. The central statement towards establishing this is the following.

Theorem 1. *For any natural numbers s and k there is a natural number $P(s, k)$ such that any graph with a path of length $P(s, k)$ has either an induced path of length s or a biclique of size k.*

We prove Theorem 1 in Section 3. Now we use this theorem to establish a corollary that the same long induced path/large biclique statement follows from a large treewidth as well.

Corollary 1. *For any natural numbers s and k there is a natural number $T(s, k)$ such that any graph of treewidth at least $T(s, k)$ either has an induced path of length s or a biclique of order k.*

Proof. It is well known (see Theorem 9 of [14]) that for each natural r there is $Y(r)$ such that if the treewidth of the given graph is at least $Y(r)$, the graph has a path of size at least r. Take $T(s, k) = Y(P(s, k))$ and apply Theorem 1. □

Theorem 2. *For fixed parameters s and k, the* NLIP-BICLIQUE *problem can be solved in a linear time.*

Proof. Let G be the input graph with n vertices. Using the linear time algorithm of Bodlaender [4], test the existence of a path of length $P(s,k)$ and find it, in case it exists.

Assume that such a path P has been found. In this case, the subgraph of G induced by the vertices of P has a biclique of size k as follows from Theorem 1. Since the size of this subgraph depends only on the parameters, the way this biclique is computed does not affect the desired runtime so, we can use the brute force.

If G does not have a path of length $P(s,q)$ then according to Corollary 1, the treewidth is at most $T(s,q)$, therefore, the biclique problem can be solved by standard techniques for graphs of bounded treewidth, say Courcelle's theorem [8]. □

3 Proof of Theorem 1

In order to prove Theorem 1, we modify it in the following way.

Theorem 3. *For every t, q, and s, there is a number $z = Z(s,t,q)$ such that every graph with a path of length at least z contains either K_t or $K_{q,q}$ or P_s as an induced subgraph.*

It is not hard to show that Theorem 1 and Theorem 3 are equivalent. Indeed, assume that Theorem 3 holds. Set $P(s,q) = Z(s,2q,q)$. It follows from Theorem 3 that a P_s-free graph with $P(s,q)$ vertices will have either a clique of size $2q$ or an induced biclique of order q. Clearly, in both cases the graph has a biclique of order q. Conversely, assume that Theorem 1 holds. Then we can just set $Z(s,t,q) = P(s, R(2,2,\max(t,q)))$, where R is the Ramsey number defined below. Thus, the equivalence between Theorem 1 and Theorem 3 has been established.

The proof of Theorem 3 consists of four stages outlined in the following four sections. On the first stage we define a class of graphs called *connecting structures*. We essentially prove that Theorem 1 holds for connecting structures, that is a sufficiently large connecting structure has either a large induced path or a large biclique. On the second stage we consider a class of graphs having a large *grid structure without an independent transversal* and we show that this is a sufficient condition for having a large biclique. On the third stage we define a class of graphs called a *bouquet* and we prove that a sufficiently large (P_s, K_t)-free graph necessarily has a large bouquet. On the final stage, we get the things together. We assume that our graph is (P_s, K_t)-free, using the third stage this immediately leads us to the conclusion that a large bouquet exists. We then show that, appropriately contracting vertices of this bouquet, we can get a large connecting structure as a subgraph. On the resulting connecting structure we consider the possibilities of a long induced path and a large biclique. Based on the assumption that the original graph is P_s-free, in both cases we infer the existence of a large grid structure without independent transversal, which in turn implies the existence of a large biclique.

Before we start the proof itself, we introduce the main tool we use in the proof, namely the fundamental result known as Ramsey's theorem, and provide a few its corollaries.

Theorem 4. *For any k, r and m, there is a number $R = R(k, r, m)$ such that in every coloring of k-subsets of an R-set with r colors there is a monochromatic m-set, i.e. a set of m elements all of whose k-subsets have the same color.*

For $k = 1$, this theorem is known as the Pigeonhole Principle. For $k = r = 2$, the number $R(2, 2, m)$ is frequently referred to as the (symmetric) Ramsey number, i.e. the minimum number such that every graph with at least $R(2, 2, m)$ vertices has either a clique of size m or an independent set of size m. In case of connected graphs, the Ramsey number admits the following generalization (see e.g. Proposition 9.4.1 in [10]).

Lemma 1. *For any t, q and s, there is a number $\ell(t, q, s)$ such that every connected graph with at least $\ell(t, q, s)$ vertices contains either K_t or $K_{1,q}$ or P_s as an induced subgraph.*

The Ramsey number also has a bipartite analog, which can be easily derived with the Pigeonhole Principle and which states that for any q, there is a number $BR(q)$ such that every bipartite graph $G = (V_1, V_2, E)$ with $|V_1| \geq BR(q)$ and $|V_2| \geq BR(q)$ has either a biclique $K_{q,q}$ or its bipartite complement. With a simple induction, this statement can be extended to multipartite graphs as follows.

Lemma 2. *For any k and q, there is a number $MR(k, q)$ such that in every k-partite graph $G = (V_1, V_2, \ldots, V_k, E)$ with $|V_i| \geq MR(k, q)$ $(i = 1, \ldots, k)$ there is a collection of subsets $U_i \subseteq V_i$ of size $|U_i| = q$ $(i = 1, \ldots, k)$ such that every pair of subsets induces either a biclique $K_{q,q}$ or its bipartite complement.*

3.1 Connecting Structures

Definition 1. *A bipartite graph $G = (A, B, E)$ is called a* connecting structure *w.r.t. A if there is an injective function f from the set $\{\{u, v\} | u, v \in A\}$ of all the unordered pairs of A to B such that $f(\{u, v\})$ is adjacent to both u and v.*

Put it differently, we call G a connecting structure w.r.t. A if for each pair $\{u, v\} \subseteq A$ we can find a vertex of B adjacent to both u and v so that different pairs are associated with different vertices.

We refer to $|A|$ as the order of the connecting structure G.

Lemma 3. *For every natural numbers s and q, there is a number $L(s, q)$ such that every connecting structure of order at least $L(s, q)$ contains either a biclique of order q or an induced path of size s.*

Proof. Let $M := \max(\lceil s/2 \rceil + 1, 2q)$ and $L(s, q) := R(3, 3, M)$ (R is the Ramsey number). Consider a connecting structure $G = (A, B, E)$ w.r.t. $A = \{a_1, a_2, \ldots, a_l\}$ where $l := L(s, q)$. We color each triple a_i, a_j, a_k $(i < j < k)$ in one of the three colors (breaking any ties between colors 1 and 2 arbitrarily):

- color 1 if a_i is adjacent to $f(\{a_j, a_k\})$,
- color 2 if a_k is adjacent to $f(\{a_i, a_j\})$,
- color 3 if neither a_i is adjacent to $f(\{a_j, a_k\})$ nor a_k is adjacent to $f(\{a_i, a_j\})$.

Then A has a subset $A' = \{a_{i_1}, \ldots, a_{i_M}\}$ of $M \geq 2q$ vertices all of whose triples have the same color. Assume that this color is 1. Then every vertex of $A_1 = \{a_{i_1}, \ldots, a_{i_q}\}$ is adjacent to every vertex of $B_1 = \{f(\{u, v\}) | u, v \in \{a_{i_q}, \ldots, a_{i_{2q}}\}\}$. The adjacency of $u \in A_1$ to $w \in B_1$ follows either from the condition of color 1 or, in case $u = a_{i_q}$ and $w = f(\{a_{i_q}, v\})$ for $v \in \{a_{i_q}, \ldots, a_{i_{2q}}\}\}$, from definition of a connecting structure. Furthermore, observe that $|A_1| = q$ and $|B_1| = \binom{q+1}{2} \geq q$ for all $q \geq 1$. It follows that the subgraph of G induced by A_1 and B_1 contains a biclique of order q.

Assume that all tripes in A' are of color 2. In this case we set $A_2 = \{a_{i_{q+1}}, \ldots, a_{i_{2q}}\}$ and $B_2 = \{f(\{u, v\}) | u, v \in \{a_{i_1}, \ldots, a_{i_{q+1}}\}\}$ and then apply regarding A_2 and B_2 the same reasoning as in the previous paragraph regarding A_1 and B_1.

Assume now that the color of all tripes in A' is 3. Consider the path

$$a_{i_1}, f(\{a_{i_1}, a_{i_2}\}), a_{i_2}, \ldots, a_{i_{M-1}}, f(\{a_{i_{M-1}}, a_{i_M}\}), a_{i_M}$$

By definition of M, the length of this path is at least s. Furthermore, observe that this path is induced. Indeed, the only possible chord is between some a_{i_x} and $f(\{a_{i_y}, a_{i_{y+1}}\})$ such that $x \neq y$ and $x \neq y + 1$. Then either $x < y$ or $x > y + 1$. In both cases such a chord is impossible according to the condition of color 3. □

3.2 Grid Structures with Large Bicliques

In a graph, a (k, t) *grid structure* is a family of $k \times t$ vertex sets $V_{i,j}$, $i = 1, \ldots, k$, $j = 1, \ldots, t$. We call $V_{i,j}$ the set in the i-th row and j-th column. A *transversal* in a grid structure is a collection of sets containing exactly one set from each row. A transversal is *independent* if no two vertices in different sets are adjacent.

Lemma 4. *For each $k \geq 2$, s and q, there is a number $C(k, s, q)$ such that any graph, having a $(k, C(k, s, q))$ grid structure with sets of size at most s and with no independent transversal, has a biclique $K_{q,q}$.*

Proof. For $k = 2$, the statement follows with a double application of the Pigeonhole Principle. In particular, we define $r := R(1, s^q, q)$, $C(2, s, q) := R(1, s^r, q)$ and consider an arbitrary collection A of r sets from the first row. Each set in the second row has a neighbor in each set of the first row, since no transversal is independent. Therefore, the family of the sets in the second row can be colored with at most s^r colors so that all sets of the same color have a common neighbor in each of the r chosen sets of collection A. By the choice of $C(2, s, q)$, one of the color classes contains a collection B of at least q sets. For each set in A, we choose a vertex which is a common neighbor for all sets in B and denote the set of r chosen vertices by U. The vertices of U can be colored with at most s^q colors so that all vertices of the same color have a common neighbor in each of the q sets of collection B. By the choice of r, U contains a color class U_1 of least q vertices. For each set in B, we choose a vertex which is a common neighbor for all vertices of U_1 and denote the set of q chosen vertices by U_2. Then U_1 and U_2 from a biclique $K_{q,q}$.

For $k > 2$, we define $C(k, s, q) := MR(k, C(2, s, q))$ (see Lemma 2 for the definition of the number MR). Since the grid structure has no independent transversal, by

Lemma 2 it must contain two rows with $C(2, s, q)$ sets in each so that the two collections of sets form a $(2, C(2, s, q))$ grid structure with no independent transversal. By the first part of the lemma, this structure contains a biclique $K_{q,q}$. □

3.3 Flowers and Bouquets

A *flower centered at* $\{a, b\}$, also called an ab-flower, consists of two distinct vertices a, b and a number of pairwise vertex disjoint induced paths connecting them. Every path in a flower will be called a *petal*. In other words, a petal is an induced path, not including a and b, such that a is adjacent to one terminal vertex of this path and b is adjacent to the other one (of course these terminal vertices may coincide in case of path of length 1). A flower with p petals will be called a p-flower. A *bouquet centered at a set of vertices* B consists of ab-flowers centered at each pair $a, b \in B$ such that no two flowers share a non-central vertex. A bouquet of p-flowers centered at a set of q vertices will be called a (p, q)-bouquet.

In this section we show that every (P_s, K_t)-free graph with a sufficiently large path contains a big bouquet with many petals in each flower. As a step toward this goal, we introduce an auxiliary structure called a *multipattern*.

A *pattern* $Z = (a, P)$ in a graph G is an induced subgraph of G consisting of a (not necessarily chordless) path P and a vertex a outside P such that a is adjacent to the first and the last vertex of P and possibly to some other vertices of P, and the subpath of P between any two consecutive neighbors of a on P is induced (i.e. chordless). If a has at least m neighbors on P, we say that the pattern $Z = (a, P)$ is m-strong.

A *multipattern* of size r in G is a sequence $(Z_1 = (a_1, P_1), \ldots, Z_r = (a_r, P_r))$ of r patterns such that for each $i > 1$, Z_i is a pattern in the subgraph of G induced by the vertices of P_{i-1}. A multipattern is m-strong if each of its patterns is m-strong.

Lemma 5. *For any natural numbers s, t, m, r, there is a number $MP(s, t, m, r)$ such that any (P_s, K_t)-free graph G with a path of length at least $MP(s, t, m, r)$ has an m-strong multipattern $(Z_1, \ldots, Z_r)$.*

Proof. We prove this lemma by induction on r. For $r = 1$, we let $MP(s, t, m, 1)$ be equal $\ell(t, 2m, s)$ (see Lemma 1 for the definition of ℓ) and consider a (P_s, K_t)-free graph G with a path P of length $MP(s, t, m, r)$. Then, by Lemma 1, the subgraph of G induced by the vertices of P must have an induced star $K_{1,2m}$. We denote by a the center of the star. At least m neighbors of a must be located either to the left or to the right of a in the order induced by P. These neighbors together with shortest (i.e. chordless) paths connecting every two consecutive neighbors and together with vertex a create an m-strong pattern in G, which proves the lemma for $r = 1$.

For $r > 1$, we inductively define $MP(s, t, m, r) := MP(s, t, M, r - 1)$, where $M = \max\{m, \ell(t, 2m, s)\}$. Then an (P_s, K_t)-free graph G with a path P of length at least $MP(s, t, m, r)$ has an M-strong multipattern $(Z_1, \ldots, Z_{r-1})$ of size $r - 1$. Since $M \geq \ell(t, 2m, s)$, the path in the pattern Z_{r-1} is of length at least $\ell(t, 2m, s)$. Therefore, as in the basis case $r = 1$, it contains an m-strong pattern Z_r. This completes the proof of the lemma. □

Lemma 6. *For any natural numbers $s, t, p, b \geq 2$, there is a number $B(s, t, p, b)$ such that any (P_s, K_t)-free graph G with a path of length at least $B(s, t, p, b)$ has a (p, b)-bouquet centered at an independent set.*

Proof. Let $B(s, t, p, b) := MP(s, t, s^2pb^2, R(2, 2, \max(t, b)))$ (R is the Ramsey number and MP is defined in Lemma 5). Then, by Lemma 5, any (P_s, K_t)-free graph G with a path of length at least $B(s, t, p, b)$ contains an s^2pb^2-strong multipattern $\mathcal{Z}$ of size $R(2, 2, \max(t, b))$. Since G is K_t-free, $\mathcal{Z}$ contains a sub-multipattern $(Z_1 = (a_1, P_1), \ldots, Z_b = (a_b, P_b))$ such that $\{a_1, \ldots, a_b\}$ is an independent set.

The neighbors of a_1 partition P_1 into vertex disjoint chordless paths each of which has at most $s - 1$ vertices (since G is P_s-free). Let us call these paths *intervals*. Vertex a_2 has neighbors in at least p of these intervals (in fact since each pattern is s^2pb^2 strong and in each interval a_2 can have at most s neighbors, the number of such neighboring intervals is at least spb^2), and each of them can be used to form a petal in the flower centered at $\{a_1, a_2\}$. This proves the lemma for $b = 2$.

For $b > 2$, assume by induction that $(Z_2 = (a_2, P_2), \ldots, Z_b = (a_b, P_b))$ contains a $(p, b-1)$-bouquet centered at vertices $\{a_2, \ldots, a_b\}$. Vertices $\{a_2, \ldots, a_b\}$ are adjacent to at most $sp(b-1)(b-2)$ vertices of the bouquet, these adjacent vertices intersect (use) at most $sp(b-1)(b-2) \leq sp(b-1)^2$ intervals of P_1. Since each interval consists of at most s vertices, vertex a_2 can have at most $s^2p(b-1)^2$ neighbors in these intervals, and since the total number of neighbors of a_2 on P_1 is at least s^2pb^2, it also has neighbors in at least p of the unused intervals (at least $s^2p(b^2 - (b-1)^2) > sp$ of unused neighbors with at most s neighbors per interval), and each of them can be used to form a petal in the flower centered at $\{a_1, a_2\}$. For $2 < i \leq b$, we assume by induction that flowers centered at $\{a_1, a_2\}, \ldots, \{a_1, a_{i-1}\}$ have been added to the bouquet. Collectively, all flowers in the bouquet use at most $sp(b-1)^2 + p(i-1) \leq sp(b-1)^2 + p(b-1) \leq spb(b-1)$ intervals of P_1. Since each interval consists of at most s vertices, vertex a_i can have at most $s^2pb(b-1)$ neighbors in these intervals, and since the total number of neighbors of a_i on P_1 is s^2pb^2, it also has neighbors in at least p of the unused intervals, and each of them can be used to form a petal in the flower centered at $\{a_1, a_i\}$. □

3.4 Proof of Theorem 3

We define $c := C(\lceil s/2 \rceil, s, q)$, $a := C(2, sc, q)$, $b := L(s, a)$, and $z := B(s, t, c, b)$ (for the definitions of numbers C, L and B see Lemmas 4, 3 and 6, respectively). Let G be a graph with a path of length z. If G contains a clique K_t or an induced path P_s, then we are done. So assume G is (K_t, P_s)-free.

By Lemma 6, G contains a (c, b)-bouquet centered at an independent set B of size b. Contract the non-central vertices of each uv-flower ($u, v \in B$) to a single vertex, called the *uv-connecting* vertex. Let X be the set of all connecting vertices. Consider a bipartite graph S with the bipartition B, X where $u \in B$ and $w \in X$ are adjacent if and only if in the graph G vertex u has a neighbour in the set of vertices contracted to w. Clearly S is a connecting structure with $f(\{u, v\})$ being the uv-connecting vertex. By Lemma 3, S has either an induced path of length s or a biclique of order a.

Assume first that S contains an induced path P of length s. This path contains at most $\lceil s/2 \rceil$ vertices of X and each of them represents a set of c petals of size at most s each.

Consider an arbitrary transversal containing one petal from each set. If this transversal is independent (i.e. no two vertices in different petals are adjacent), then by replacing the vertices of X in P by the respective petals of the transversal, we obtain an induced path of length at least s in the original graph G, which is impossible. Therefore, each transversal has at least one edge and hence by Lemma 4 G has a biclique of size q.

Suppose now that the connecting structure S has a biclique of order a. Each connecting vertex of this biclique represents a set of c petals of size at most s each. Therefore, this biclique represents a $(2, a)$ grid structure of G with sets of size at most sc and with no independent transversal. Therefore, by Lemma 4, G has a biclique of size q.

4 Directions of Future Research

In this paper, we have shown that computing a biclique of order k in a P_s-free graph is fixed-parameter tractable when parameterized by k and s. The main ingredient of the proposed method is Theorem 1 that establishes connection between the Biclique problem and a W[1]-hard problem Induced Path. This might give a hope of a possibility of establishing W[1]-hardness of the Biclique problem by a reduction from the Induced path problem. However, it is not clear how such a reduction would work in the presence of a *large* biclique.

Intuitively, Biclique problem is 'similar' to the Clique problem (but much more resistant to the attempts of proving W[1]-hardness). Does this similarity preserve in the case of P_s-free graphs? In particular, what is the complexity of k-Clique problem in P_s-free graphs where k and s as parameters? In fact it would be a strong result even if this problem shown FPT on k with the power of polynomial depending on s: it will show, for instance, that the Clique problem is FPT for P_5-free graphs, where it is known to be NP-hard by a reduction from the Independent set problem on graphs with large girth [23]. Dániel Marx suggested that an interesting intermediate problem between Biclique and Clique is the Tripartite Clique, i.e. finding out if the given P_s-free graph has a complete 3-partite subgraph with k vertices in each part. Although these problems are closely related to the result proposed in this paper, it is not clear how Theorem 1 can help in their resolution: P_s-free graphs of large treewidth cannot be guaranteed to have a large clique, not even a large tripartite clique, because of the failure of such potential theorems on bipartite graphs. Therefore, if any of these problems is FPT, new methods would be required to establish this.

Finally, it is interesting to see how Theorem 1 can be modified and extended. In particular, assume that the given graph *does not* have a large biclique. In this case if the largeness condition is a *large average degree* then the consequences are very strong: as shown in [19], the considered graph will have an induced subdivision of *any* graph. In this paper, we prove that if the largeness condition is a *long path*, then the absence of a large biclique implies 'just' a long induced path. Can we claim more than that? What if the 'long path' condition is replaced by a stronger 'large treewidth' assumption?

Acknowledgments. Research of the first two authors was supported by the Centre for Discrete Mathematics and Its Applications (DIMAP) at the University of Warwick. The second author also acknowledges a support from EPSRC, grant EP/I01795X/1.

Part of this research was done when the third author visited the Institute of Informatics of the Humboldt University of Berlin (hosted by Martin Grohe and Dániel Marx) and the Department of Informatics of the University of Bergen (hosted by Fedor Fomin). Both of the visits took place in August 2011. We would like to thank Dániel Marx for interesting conversations regarding possible further research related to the proposed result. We would also like to thank anonymous reviewers for noticing a number of minor mistakes in the preliminary version of this paper.

References

1. Arbib, C., Mosca, R.: Polynomial algorithms for special cases of the balanced complete bipartite subgraph problem. J. Combin. Math. Combin. Comput. 39, 3–22 (1999)
2. Bascó, G., Tuza, Z.: A characterization of graphs without long induced paths. J. of Graph Theory 14, 455–464 (1990)
3. Binkele-Raible, D., Fernau, H., Gaspers, S., Liedloff, M.: Exact exponential-time algorithms for finding bicliques. Inf. Process. Lett. 111(2), 64–67 (2010)
4. Bodlaender, H.L.: On linear time minor tests with depth-first search. J. Algorithms 14(1), 1–23 (1993)
5. Broersma, H., Golovach, P.A., Paulusma, D., Song, J.: Updating the complexity status of coloring graphs without a fixed induced linear forest. Theor. Comput. Sci. 414(1), 9–19 (2012)
6. Bulatov, A.A., Marx, D.: Constraint Satisfaction Parameterized by Solution Size. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 424–436. Springer, Heidelberg (2011)
7. Chen, Y., Thurley, M., Weyer, M.: Understanding the Complexity of Induced Subgraph Isomorphisms. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 587–596. Springer, Heidelberg (2008)
8. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width. In: Hromkovič, J., Sýkora, O. (eds.) WG 1998. LNCS, vol. 1517, pp. 1–16. Springer, Heidelberg (1998)
9. Demaine, E., Gutin, G.Z., Marx, D., Stege, U.: 07281 open problems – structure theory and FPT algorithmcs for graphs, digraphs and hypergraphs. In: Demaine, E., Gutin, G.Z., Marx, D., Stege, U. (eds.) Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs, Dagstuhl, Germany. Dagstuhl Seminar Proceedings, vol. (07281), Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
10. Diestel, R.: Graph Theory, 3rd edn. Springer (2005)
11. Dong, J.: Some results on graphs without long induced paths. J. of Graph Theory 22, 23–28 (1996)
12. Feige, U., Kogan, S.: Hardness of approximation of the balanced complete bipartite subgraph problem. Technical Report MCS04-04, Weizmann Institute of Science (2004)
13. Fellows, M., Gaspers, S., Rosamond, F.: Multivariate complexity theory. In: Blum, E.K., Aho, A.V. (eds.) Computer Science: The Hardware, Software and Heart of It, pp. 269–293. Springer (2011)
14. Fellows, M.R., Langston, M.A.: On search, decision and the efficiency of polynomial-time algorithms (extended abstract). In: STOC, pp. 501–512 (1989)
15. Golovach, P.A., Paulusma, D., Song, J.: Coloring Graphs without Short Cycles and Long Induced Paths. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 193–204. Springer, Heidelberg (2011)

16. Johnson, D.S.: The NP-completeness column: An ongoing guide. J. Algorithms 8(3), 438–448 (1987)
17. Korobitsyn, D.: On the complexity of determining the domination number in monogenic classes of graphs. Diskretnaya Matematika 2(3), 90–96 (1990) (in Russian)
18. Král', D., Kratochvíl, J., Tuza, Z., Woeginger, G.J.: Complexity of Coloring Graphs without Forbidden Induced Subgraphs. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 254–262. Springer, Heidelberg (2001)
19. Kühn, D., Osthus, D.: Induced subdivisions in $K_{S,S}$-free graphs of large average degree. Combinatorica 24(2), 287–304 (2004)
20. Le, V.B., Randerath, B., Schiermeyer, I.: On the complexity of 4-coloring graphs without long induced paths. Theor. Comput. Sci. 389(1-2), 330–335 (2007)
21. Lozin, V.V., Mosca, R.: Maximum independent sets in subclasses of P_5-free graphs. Inf. Process. Lett. 109(6), 319–324 (2009)
22. Lozin, V.V., Rautenbach, D.: Some results on graphs without long induced paths. Inf. Process. Lett. 88(4), 167–171 (2003)
23. Murphy, O.J.: Computing independent sets in graphs with large girth. Discrete Applied Mathematics 35(2), 167–170 (1992)
24. Randerath, B., Schiermeyer, I.: 3-colorability in P for P_6-free graphs. Discrete Applied Mathematics 136(2-3), 299–313 (2004)
25. Woeginger, G.J., Sgall, J.: The complexity of coloring graphs without long induced paths. Acta Cybernetica 15(1), 107–117 (2001)

Induced Disjoint Paths in AT-Free Graphs*

Petr A. Golovach[1], Daniël Paulusma[1], and Erik Jan van Leeuwen[2]

[1] School of Engineering and Computing Sciences, Durham University
Science Laboratories, South Road, Durham DH1 3LE, UK
{petr.golovach,daniel.paulusma}@durham.ac.uk

[2] Dept. Computer and System Sciences, University of Rome "La Sapienza"
Via Ariosto 25, 00185 Roma, Italy
E.J.van.Leeuwen@ii.uib.no

Abstract. Paths $P_1, \ldots, P_k$ in a graph $G = (V, E)$ are said to be mutually induced if for any $1 \leq i < j \leq k$, P_i and P_j have neither common vertices nor adjacent vertices (except perhaps their end-vertices). The Induced Disjoint Paths problem is to test whether a graph G with k pairs of specified vertices (s_i, t_i) contains k mutually induced paths P_i such that P_i connects s_i and t_i for $i = 1, \ldots, k$. This problem is known to be NP-complete already for $k = 2$. We prove that it can be solved in polynomial time for AT-free graphs even when k is part of the input. As a consequence, the problem of testing whether a given AT-free graph contains some graph H as an induced topological minor admits a polynomial-time algorithm, as long as H is fixed; we show that such an algorithm is essentially optimal by proving that the problem is W[1]-hard, even on a subclass of AT-free graphs, namely cobipartite graphs, when parameterized by $|V_H|$. We also show that the problems k-in-a-Path and k-in-a-Tree can be solved in polynomial time, even when k is part of the input. These problems are to test whether a graph contains an induced path or induced tree, respectively, spanning k given vertices.

1 Introduction

We study the induced version of the well-known Disjoint Paths problem for the class of AT-free graphs. Before we define this graph class and explain our results, we first introduce the problem, survey the known results, and mention its applications.

The Induced Disjoint Paths problem takes as input a pair (G, S), where G is a graph and $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ is a set of pairs of specified vertices of a graph G, which we call *terminals*, and tests whether there exist (not necessarily induced) paths $P_1, \ldots, P_k$ in G that satisfy the following conditions:

i) any distinct P_i, P_j may only share vertices that are ends of both paths.
ii) for all $i \neq j$, no inner vertex u of P_i is adjacent to a vertex v of some P_j, except when v is an end-vertex of both P_i and P_j.
iii) for all i, P_i has ends s_i and t_i.

* This work is supported by EPSRC (EP/G043434/1), Royal Society (JP100692), and ERC StG project PAAl no. 259515.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 153–164, 2012.

As can be seen from this definition, we relax the conditions imposed on the end-vertices (terminals) of the paths P_i. We explain why we do this later.

The Induced Disjoint Paths problem is a variant of the well-known Disjoint Paths problem. The latter problem is to test whether a graph G with k pairs of specified vertices (s_i, t_i) contains a set of k mutually vertex-disjoint paths $P_1, \ldots, P_k$. Note that if we subdivide each edge of an input graph of the Disjoint Paths problem, then we obtain an equivalent instance of the Induced Disjoint Paths problem. However, we cannot do this on graph classes that are not closed under the operation of edge subdivision, and AT-free graphs form an example of such a class.

The Disjoint Paths problem is one of the problems in Karp's list of NP-compete problems [13]. If k is a *fixed* integer, i.e., not part of the input, it can be solved in $O(n^3)$ time for n-vertex graphs, as shown by Robertson and Seymour [23]. In contrast, the Induced Disjoint Paths problem is NP-complete even if $k = 2$, as shown by Fellows [9] and Bienstock [2]. Hence graph classes for which the k-Induced Disjoint Paths problem may be tractable have been identified. Below, we briefly survey results for classes studied so far.

For planar graphs, Induced Disjoint Paths stays NP-complete; we can subdivide each edge of a planar input graph of Disjoint Paths to obtain a planar input graph of Induced Disjoint Paths and use the result that Disjoint Paths is NP-complete for planar graphs, as shown by Lynch [20]. However, Kobayashi and Kawarabayashi [15] presented an algorithm that solves Induced Disjoint Paths on planar graphs that runs in linear time for any fixed k. For claw-free graphs, Induced Disjoint Paths stays NP-complete as well. This is shown by Fiala et al. [10], even for line graphs which form a subclass of claw-free graphs. They also showed that the problem can be solved in polynomial time if k is fixed [10]. Recently, we improved this result by proving that Induced Disjoint Paths is fixed-parameter tractable for claw-free graphs when parameterized by k [11]. For chordal graphs, Induced Disjoint Paths is polynomial-time solvable, as shown by Belmonte et al. [1].

Algorithms that solve the Induced Disjoint Paths problem for graph classes have numerous applications. For example, they are used as a subroutine in algorithms that detect induced containment relations, as explained below.

A graph G contains a graph H as a *topological (induced) minor* if G contains an (induced) subgraph that is isomorphic to a subdivision of H, i.e., to a graph obtained from H by a number of edge subdivisions. The problems that are to test whether a given graph contains some fixed graph H as a topological minor or induced topological minor are called H-Topological Minor and H-Induced Topological Minor. Robertson and Seymour [23] showed that H-Topological Minor can be solved in polynomial time for any fixed graph H. Grohe et al. [12] improved this result to cubic time. The complexity classification for H-Induced Topological Minor is wide open, although both polynomial-time and NP-complete cases are known, as shown by Lévêque et al. [19].

Due to our relaxation of conditions i) and ii) for the terminals, we can use an algorithm for Induced Disjoint Paths as a subroutine in an algorithm for H-Induced Topological Minor. Hence, for any fixed graph H, we solve H-Induced Topological Minor in polynomial time on any graph class for which we can solve the Induced Disjoint Paths problem in polynomial time for fixed k, such as for the classes of planar graphs, claw-free graphs, or chordal graphs. Note that this approach cannot be used for general graphs, as Induced Disjoint Paths is already NP-complete when $k = 2$ [2,9].

The related problem of detecting an induced subgraph containing a set of k specified vertices (called terminals as well) has also been extensively studied. This is in particular the case if the subgraph is required to be a tree, cycle, or path. Then the problem is called k-in-a-Tree, k-in-a-Cycle, and k-in-a-Path, respectively. Derhy and Picouleau [5] showed that k-in-a-Tree is NP-complete when k is part of the input, whereas Chudnovsky and Seymour [4] proved that the 3-in-a-Tree problem is polynomial-time solvable. The complexity of k-in-a-Tree is open for all fixed $k \geq 4$. The problems 2-in-a-Cycle and 3-in-a-Path are NP-complete, which follows from the aforementioned results of Fellows [9] and Bienstock [2]. Again, due to our relaxation of conditions i) and ii), we can use an algorithm for Induced Disjoint Paths as a subroutine for solving these three problems. This leads to polynomial-time algorithms for solving them for any fixed k on any graph class for which we can solve the Induced Disjoint Paths problem in polynomial time for fixed k.

We consider the class of *asteroidal triple-free* graphs, also known as *AT-free* graphs. An *asteroidal triple* is a set of three mutually non-adjacent vertices such that each two of them are joined by a path that avoids the neighborhood of the third, and AT-free graphs are exactly those graphs that contain no such triple. AT-free graphs, defined fifty years ago by Lekkerkerker and Boland [18], are well studied in the literature with respect to a variety of graph problems, such as finding minimum dominating sets, minimum feedback vertex sets, longest paths, and coloring [14,16,17,24]. Moreover, the class of AT-free graphs contains many well-known classes, such as cocomparability graphs, cographs, interval graphs, permutation graphs, and trapezoid graphs (cf. [6]).

Our Results. We prove that the Induced Disjoint Paths problem is polynomial-time solvable for AT-free graphs, even if k is part of the input. This is somehow surprising, as the related Disjoint Paths problem is NP-complete already for interval graphs [22]. Usually, the induced variant of a containment relation problem is computationally just as hard or even harder than its non-induced variant, as described above for general graphs, planar graphs, and claw-free graphs.

We explain our algorithm for solving Induced Disjoint Paths in Section 3. We first provide a thorough exploration of the structure of AT-free graphs, in particular in relation to the Induced Disjoint Paths problem. We then use these structural results for a complex dynamic programming algorithm. We should note that our approach is substantially different from the approach for chordal graphs by Belmonte et al. [1], as it appears that their tree-decomposition-based approach cannot work for AT-free graphs.

In Section 4, we determine to which extent we can use our algorithm for solving Induced Disjoint Paths for detecting induced topological minors in AT-free graphs. First, we consider the anchored version of H-Induced Topological Minor. This variant is to test whether a graph G contains a graph H as an induced topological minor in such a way that the vertices of H are mapped to specified vertices of G. We show that the anchored version of H-Induced Topological Minor can be solved in polynomial time on AT-free graphs even when H is an arbitrary graph that is part of the input. This result cannot be generalized to the original, non-anchored version of this problem. However, our result for Induced Disjoint Paths still implies a polynomial-time algorithm for H-Induced Topological Minor on AT-free graphs if H is fixed. This may be the best we can hope for, as we prove that Induced Topological Minor is W[1]-hard when parameterized by $|V_H|$, even on cobipartite graphs (a subclass of AT-free graphs).

In Section 5, we consider the problems k-in-a-Tree and k-in-a-Path; note that k-in-a-Cycle is trivial on AT-free graphs, because AT-free graphs do not contain induced cycles on six or more vertices. As we explained, our result for Induced Disjoint Paths implies that k-in-a-Tree and k-in-a-Path can be solved in polynomial time for AT-free graphs, that is, provided that k is fixed, as we may need to consider $k!$ ordering of the terminals. In contrast to the H-Induced Topological Minor problem, however, we can give a direct approach showing that both k-in-a-Tree and k-in-a-Path can be solved in polynomial time for AT-free graphs even when k is part of the input.

2 Preliminaries

Let $G = (V, E)$ be a graph. Throughout the paper, we consider only finite, undirected graphs without multiple edges and self-loops. The graph $G[S]$ denotes the subgraph of G induced by S. We denote the (open) neighborhood of a vertex u by $N(u) = \{v \mid uv \in E\}$ and its closed neighborhood by $N[u] = N(u) \cup \{u\}$. We denote the (open) neighborhood of a set $U \subseteq V$ by $N(U) = \{v \in V \setminus U \mid uv \in E$ for some $u \in U\}$ and its closed neighborhood by $N[U] = N(U) \cup U$. We let $d(u) = |N(u)|$ denote the *degree* of a vertex u. Whenever it is not clear from the context, we may add an extra subscript $_G$ to these notations. A set of vertices $U \subseteq V$ of a graph $G = (V, E)$ is a *dominating set* if $u \in U$ or $u \in N(U)$ for each $u \in V$. A pair of vertices $\{x, y\}$ is a *dominating pair* if the vertex set of every x, y-path dominates G. A path P *dominates* a vertex u if $u \in N[V_P]$. A path P dominates a vertex set U if it dominates each $u \in U$. Corneil, Olariu and Stewart [6,7] proved that every connected AT-free graph has a dominating pair and that such a pair can be found in linear time.

A collection of paths $P_1, \ldots, P_k$ that satisfy the aforementioned conditions i)–iii) for a given graph G with a set S of k terminals pairs (s_i, t_i) is called a *solution* for (G, S). Note that these paths do not have to be induced paths of G. However, as we can take shortcuts if necessary, we may without loss of generality assume that they are induced. As a matter of fact, this is convenient for algorithmic purposes and from now on we assume that in a solution

iv) each P_i is an induced path.

We allow a terminal to be in more than one terminal pair, but no two terminal pairs may coincide and the terminals of each pair may not be the same, i.e., we assume

v) $\{s_i, t_i\} \neq \{s_j, t_j\}$ for all $i \neq j$ and $s_i \neq t_i$ for all i .

We can assume condition v), as our algorithm for detecting induced topological minors, which uses our algorithm for Induced Disjoint Paths as a subroutine, does not require two terminal pairs to coincide, as we shall see. It may happen that in this application two terminals of the same pair are the same, but we can easily work around this. Moreover, our algorithm for Induced Disjoint Paths can be modified to deal with terminal pairs (s_i, t_i) with $s_i = t_i$, but we refrain from doing so to avoid any further technicalities. Throughout the paper, we let $T = \bigcup_{i=1}^{k} \{s_i, t_i\}$ denote the set of terminals, and $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ the set of terminal pairs.

3 Induced Disjoint Paths

In this section we present our polynomial-time algorithm for the Induced Disjoint Paths problem. Our algorithm has as input a pair (G, S), where G is an AT-free graph and S is a set of terminal pairs, and consists of the following three phases.

Phase 1. Preprocess (G, S) to derive a number of convenient properties. For instance, afterwards two terminals of the same pair are non-adjacent.

Phase 2. Derive a number of structural properties of (G, S). The algorithm constructs an auxiliary graph H, which is obtained from the subgraph of G induced by the terminal vertices by adding a path of length two between each pair of terminals. It then checks whether H satisfies some necessary conditions, such as being AT-free and being an anchored topological minor of G. The latter condition is also shown to be sufficient and demands the construction of another auxiliary graph in order to describe how induced paths connecting terminal pairs may interfere with each other.

Phase 3. Perform dynamic programming using the information gathered in Phases 1-2.

3.1 Phase 1: Preprocessing

Let (G, S) be an instance of Induced Disjoint Paths, where $G = (V, E)$ is an AT-free graph on n vertices with a set S of k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. Recall that $T = \bigcup_{i=1}^{k} \{s_i, t_i\}$ denotes the set of terminals.

We need the following preprocessing steps: we first apply Step 1, then Step 2, and then Step 3, where we perform each step as long as possible. In the below, $M(u)$ denotes the set of all terminals that form a terminal pair with vertex u.

Step 1. For all $u, v \in T$ with $uv \in E$, remove all common neighbors of u and v that are not terminals from G, i.e., remove all vertices of $N(u) \cap N(v) \cap (V \setminus T)$.

Step 2. For all $u \in T$ with $M(u) \subseteq N(u)$, remove u and all of its neighbors that are not terminals from G. Also remove all (s_j, t_j) with $u \in \{s_j, t_j\}$ from S.

Step 3. For all (s_i, t_i) with $s_i t_i \in E$, remove (s_i, t_i) from S.

We have the following lemma. The proof of this lemma and of the other lemmas in this paper has been omitted due to space restrictions.

Lemma 1. *Applying Steps* 1–3 *takes polynomial time and results in a new instance* (G', S')*, where* G' *is an induced (and hence AT-free) subgraph of* G *and* $S' \subseteq S$*, such that* (G', S') *is a* `Yes`*-instance of* Induced Disjoint Paths *if and only if* (G, S) *is a* `Yes`*-instance.*

For convenience we denote the obtained instance by (G, S) as well, and we also assume that $|S| = k$.

For $i = 1, \ldots, k$, let G_i denote the subgraph obtained from G after removing all terminal vertices not equal to s_i or t_i, together with all of their neighbors not equal to s_i or t_i (should s_i or t_i be adjacent to a terminal from some other pair), i.e., G_i is the subgraph of G induced by $(V \setminus (\cup_{v \in T \setminus \{s_i, t_i\}} N[v])) \cup \{s_i, t_i\}$. The following lemma is straightforward to see.

Lemma 2. *If* s_i *and* t_i *are in different connected components of* G_i *for some* $1 \leq i \leq k$*, then* (G, S) *is a* `No`*-instance of* Induced Disjoint Paths.

Hence, we can add the following preprocessing step, which we can perform in polynomial time.

Step 4. If some s_i and t_i are in two different connected components of G_i, then return `No`.

Summarizing, applying Steps 1–4 takes polynomial time and results in the following additional conditions for our instance:

vi) the terminals of each pair (s_i, t_i) are not adjacent;
vii) the terminals of each pair (s_i, t_i) are in the same connected component of G_i.

3.2 Phase 2: Obtaining Structural Results

We need the following terminology. Let G be a graph and let $uw \in E$. The *edge subdivision* of uw removes uw and adds a new vertex v with edges uv and vw. A graph G' is a *subdivision* of G if G' can be obtained from G by a sequence of edge subdivisions. A graph H is an *induced topological minor* of G, if G has an induced subgraph that is isomorphic to a subdivision of H.

Let G be a graph in which we specify r distinct vertices $p_1, \dots, p_r$. Let H be a graph in which we specify r distinct vertices $q_1, \dots, q_r$. Then G contains H as an induced topological minor *anchored* in $(p_1, q_1), \dots, (p_k, q_k)$ if G contains an induced subgraph isomorphic to a subdivision of H such that the isomorphism maps p_i to q_i for $i = 1, \dots, r$. The graphs G and H may have common vertices. If $p_i = q_i$ for $i = 1, \dots, r$, we speak of "being anchored in $p_1, \dots, p_k$" instead.

Now let G be an AT-free graph with a set $S = \{(s_1, t_1), \dots, (s_k, t_k)\}$ of terminal pairs satisfying conditions i)–vii). In $G[T]$, there is no edge between any two terminals of the same pair due to condition vi). We modify $G[T]$ by joining the terminals of each pair (s_i, t_i) by a path P_i of length two, i.e., for each pair (s_i, t_i) we introduce a new vertex that we make adjacent only to s_i and t_i. We denote the resulting graph by H. Note that $G[T]$ is an induced subgraph of H. The inner vertex of each P_i is called a *path-vertex*, and the two edges of each P_i are called *path-edges*.

The following lemma is straightforward to see.

Lemma 3. *The pair* (G, S) *is a* `Yes`*-instance of* Induced Disjoint Paths *if and only if* G *contains* H *as an induced topological minor anchored in the terminals of* T.

Lemma 3 immediately implies the next lemma.

Lemma 4. *If* (G, S) *is a* `Yes`*-instance of* Induced Disjoint Paths, *then* H *is AT-free.*

Lemma 4 yields the following step.

Step 5. If H is not an AT-free graph, then return `No`.

From now on, we assume that H is AT-free. Due to Step 5, we have the following lemma.

Lemma 5. *Every vertex* $u \in T$ *is included in at most five terminal pairs.*

Let $H_1, \dots, H_r$ be the connected components of H. We observe that the terminals of each pair are in the same connected component of H due to the paths P_i. Hence, we can define the set $S_i \subseteq S$ of terminal pairs in a connected component H_i. We write $T_i = V_{H_i} \cap T$ to denote the set of terminals in H_i.

Lemma 6. *Each* H_i *has a dominating pair* $\{x_i, y_i\}$ *with* $x_i, y_i \in T_i$. *Moreover, such a dominating pair can be found in linear time.*

For the dominating pairs $\{x_i, y_i\}$ found by Lemma 6, we compute in linear time a shortest x_i, y_i-path D_i in H_i for each $i = 1, \dots, r$. The next two lemmas show a number of properties of these paths D_i.

Lemma 7. *Each* D_i *contains at least one of the terminals of every pair in* S_i.

Lemma 8. *Each vertex of* D_i *is adjacent to at most five path-vertices of* H_i *that are not on* D_i *and to at most two terminals that are not on* D_i.

By Lemma 3, we must check whether G contains H as an induced topological minor anchored in the terminals of T. In other words, we must check whether G contains an induced subgraph that is isomorphic to a subdivision H' of H, such that the isomorphism maps every terminal of G to the same terminal of H. Because $G[T]$ is an induced subgraph of H, such a subdivision H' contains no subdivided edges of H between two terminals of the same pair. Hence, H' is obtained from H after subdividing a number of path-edges one or more times (note that $H' = H$ is possible). For this purpose, we need the following lemma.

Lemma 9. *Let H_i' be an AT-free graph obtained from H_i by subdividing a number of path-edges one or more times. Let P_j' and D_i' be the resulting paths obtained from the paths P_j and D_i, respectively. Then,*

a) the length of each path P_j' that is not a subpath of D_i' is at most three (implying that every internal vertex of P_j' is adjacent to at least one of s_j, t_j);
b) D_i' dominates all but at most two vertices of H_i'.

Condition vii) tells us that the terminals s_i and t_i are in the same connected component of the graph G_i for $i = 1, \ldots, k$. Two terminal pairs (s_i, t_i) and (s_j, t_j) are *interfering* if there is an induced s_i, t_i-path Q_i in G_i and an induced s_j, t_j-path Q_j in G_j, respectively, such that Q_i and Q_j are not mutually induced. We say that there is *interference* between two sets of terminal pairs S_i and S_j if a terminal pair from S_i and a terminal pair from S_j are interfering.

Lemma 10. *Let (s_i, t_i) and (s_j, t_j) be interfering terminal pairs from two different connected components of H. Let Q_i and Q_j be induced s_i, t_i- and s_j, t_j-paths in G_i and G_j, respectively, such that Q_i and Q_j are not mutually induced. Then Q_i and Q_j are vertex disjoint. Moreover, for any edge $uv \in E_G$ with $u \in V_{Q_i}$ and $v \in V_{Q_j}$, it holds that $u \in N(s_i) \cup N(t_i)$ and $v \in N(s_j) \cup N(t_j)$.*

Lemma 10 implies the following lemma.

Lemma 11. *It is possible to check in polynomial time whether two terminal pairs (s_i, t_i) and (s_j, t_j) from different connected components of H are interfering.*

Lemma 11 enables us to construct in polynomial time an auxiliary graph I with vertices $1, \ldots, r$ and edges ij if and only if there is interference between S_i and S_j. This leads to the following lemma.

Lemma 12. *The graph I is a disjoint union of paths.*

Let $I_1, \ldots, I_l$ be the connected components of I, and let $J_1, \ldots, J_l$ be their vertex sets respectively. For $h = 1, \ldots, l$, we define $X_h = \{(s_j, t_j) \mid (s_j, t_j) \in \cup_{p \in J_i} S_p\}$, and we let G_h^* be the graph obtained from G by removing all vertices of the closed neighborhoods of all the terminals that are not included in X_h. Lemma 13 shows how G is related to the graphs G_h^*.

Lemma 13. *The instance (G, S) is a* `Yes`*-instance of* Induced Disjoint Paths *if and only if (G_h^*, X_h) is a* `Yes`*-instance for all $1 \leq h \leq l$.*

Lemma 13 gives us the following step.

Step 6. If I is disconnected, then solve Induced Disjoint Paths for each (G^*_h, X_h). Return `Yes` if the answer is `Yes` for all of these instances and return `No` otherwise.

By Step 6, we may assume that I is connected. Then, by Lemma 12, we may assume that I is a path. This leads to the following new condition:

viii) there is interference between two sets S_i and S_j for some $1 \leq i < j \leq r$ if and only if $j = i + 1$.

Lemma 10 gives us some structural information about interfering terminal pairs. We are now ready to be a bit more precise, which is necessary for our algorithm. For $i = 1, \ldots, r-1$, let W_i be the set of all vertices $u \in V_G$, such that there is an edge $uv \in E$ with the following property: there are interfering terminal pairs (s_p, t_p) and (s_q, t_q) in S_i and S_{i+1}, respectively, such that G_p has an induced s_p, t_p-path containing u and G_q has an induced s_q, t_q-path containing v. Using this definition we can state the next lemma.

Lemma 14. *For $i = 1, \ldots, r-1$, there is a set of terminals $Z_i \subseteq T_i$ such that $W_i \subseteq N(Z_i)$ and Z_i contains either one terminal or two adjacent terminals.*

Using Lemmas 10 and 14, we obtain the following result.

Lemma 15. *The sets $Z_1, \ldots, Z_{r-1}$ can be found in polynomial time.*

3.3 Phase 3: Dynamic Programming

Now we are ready to give a dynamic-programming algorithm for Induced Disjoint Paths. For simplicity, we solve the decision problem here, i.e., we only check for the existence of paths, but the algorithm can be modified to get the paths themselves (if they exist). Due the space restrictions we only sketch the main ideas used by the algorithm.

Our algorithm is based on the following separation property of AT-free graphs.

Lemma 16. *Let u, v be two vertices and let P be an induced u, v-path of length at least four in an AT-free graph G. Let G' be the subgraph obtained from G after removing $N[V_P \setminus \{u, v\}]$. If G_1 and G_2 are connected components of G' containing a neighbor of u and v in G, respectively, then $G_1 \neq G_2$.*

Let G be a connected AT-free graph and let $S = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a set of terminal pairs. We assume that this instance of Induced Disjoint Paths satisfies conditions i)–viii). We also assume that the auxiliary graph H with the connected components $H_1, \ldots, H_r$ is given, together with the sets of terminal pairs $S_1, \ldots, S_r$ and the sets of terminals $T_1, \ldots, T_r$. Then for $i \in \{1, \ldots, r-1\}$, the sets $Z_i \subseteq T_i$ are constructed using Lemmas 14 and 15. We let $Z_0 = \emptyset$.

The dynamic programming consists of two stages. First, we construct the subroutine $\texttt{Component}(i, X, Y)$ that for each $i \in \{1, \ldots, r\}$ and for any two sets $X \subseteq N(Z_{i-1})$, $Y \subseteq N(Z_i)$ of size at most 10 each, solves Induced Disjoint Paths for the graph $F_i = G[V \setminus (\cup_{j \in \{1, \ldots r\}, j \neq i} \cup_{u \in T_j} N[u])]$ with the set of terminal pairs S_i with the following conditions:

a) the paths from a solution are not adjacent to the vertices of X,
b) the set of non-terminal vertices from $N(Z_i)$ used by the paths in a solution is a subset of Y.

To construct $\texttt{Component}(i, X, Y)$, we consider H_i. Recall that this graph has dominating path D_i that joins the terminals x_i and y_i. In our algorithm we are "tracing" an x_i, y_i-path D'_i in F_i that is a subdivision of D_i. Our algorithm is based on the fact that by Lemma 16, we can do this by keeping only the last at most five vertices of the already constructed subpath originating in x_i. Also recall that, by Lemma 7, at least one vertex of each pair of terminals in F_i should be in D'_i. Moreover, by Lemma 8, we are interested in only a bounded number of vertices adjacent to the vertices of D'_i, and by Lemma 9, the paths outside D'_i are short and D'_i dominates almost all vertices of these paths.

In the second stage we "glue" solutions for Induced Disjoint Paths for F_i provided by $\texttt{Component}(i, X, Y)$. Our algorithm solves, for each $i \in \{1, \ldots, r\}$ and $Y \subseteq N(Z_i)$ of size at most ten, Induced Disjoint Paths for the graph $F'_i = G[V \setminus (\cup_{j \in \{i+1, \ldots r\}} \cup_{u \in T_j} N[u])]$ with the set of terminal pairs $S'_i = \cup_{j=1}^{i} S_j$ with the following additional condition: the set of non-terminal vertices from $N(Z_i)$ used by the paths in a solution is a subset of Y. Observe that by Lemma 5, any vertex $u \in T$ is included in at most five terminal pairs. Hence, we consider only sets Y of size at most ten. We solve this problem consecutively for $i = 1, \ldots, r$. Clearly, if we have a solution for $i = r$, then we have the $\texttt{Yes}$-answer for Induced Disjoint Paths for G and S. If for some $i \in \{1, \ldots, r\}$ we get a $\texttt{No}$-answer, then we stop and give a $\texttt{No}$-answer for the original instance. The algorithm runs in polynomial time and this implies our main result.

Theorem 1. *The* Induced Disjoint Paths *problem can be solved in polynomial time for AT-free graphs.*

4 Induced Topological Minors

The Induced Topological Minor problem is to test whether a given graph G contains a given graph H as an induced topological minor. The Anchored Induced Topological Minor problem is to test whether a given graph G has an induced subgraph that is a subdivision of a given graph H, such that H is anchored in $V_H \subseteq V_G$.

It is easy to reduce Anchored Induced Topological Minor to Induced Disjoint Paths on G: for each edge $uv \in E_H$, we construct the pair of terminals (u, v). Hence, Theorem 1 immediately implies the following.

Corollary 1. *The* Anchored Induced Topological Minor *problem can be solved in polynomial time for AT-free graphs.*

If H is a *fixed* graph, i.e. not part of the input, then Induced Topological Minor can be solved in polynomial time, as we mentioned in Section 1.

Corollary 2. *The* Induced Topological Minor *problem can be solved in time $n^{k+O(1)}$ for pairs (G, H) where G is an n-vertex AT-free graph and H is a k-vertex graph.*

However, if H is a part of the input or if we parameterize the problem by the size of H, then we can show hardness even for a subclass of AT-free graphs. A graph is *cobipartite* if its vertex set can be partitioned into two cliques. Clearly, any cobipartite graph is AT-free.

Theorem 2. *The* Induced Topological Minor *problem is* NP*-complete for cobipartite graphs, and* W[1]*-hard for cobipartite graphs parameterized by $|V_H|$.*

5 Concluding Remarks

We have presented a polynomial-time algorithm that solves Induced Disjoint Paths for AT-free graphs, and applied this algorithm on the H-Induced Topological Minor problem for this graph class. By similar arguments we obtain the following theorem.

Theorem 3. *The k-in-a-Tree problem and the k-in-a-Path problem can be solved in polynomial time for AT-free graphs.*

Motivated by our application on testing for induced topological minors, we assumed that all terminal pairs in an instance (G, S) of Induced Disjoint Paths are distinct. For general graphs, we can easily drop this assumption by replacing a vertex u representing $\ell \geq 2$ terminals by ℓ new mutually non-adjacent vertices, each connected to all neighbors of u via subdivided edges. This yields an equivalent instance of Induced Disjoint Paths, in which all terminal pairs are distinct. However, we cannot apply this reduction for AT-free graphs, because it may create asteroidal triples. Hence the complexity of this more general problem remains an open question. If we consider the special case in which all terminal pairs coincide, i.e. in which $(s_1, t_1) = \cdots = (s_k, t_k)$, we note that this problem is already NP-compete for $k = 2$ [2,9] and solvable in $O(n^2)$ time on n-vertex planar graphs for arbitrary k [21]. However, we can solve this case in polynomial time for AT-free graphs.

References

1. Belmonte, R., Golovach, P.A., Heggernes, P.: van't Hof, P., Kaminski, M., Paulusma, D.: Detecting patterns in chordal graphs (preprint)
2. Bienstock, D.: On the complexity of testing for odd holes and induced odd paths. Disc. Math. 90, 85–92 (1991); See also Corrigendum. Disc. Math. 102, 109 (1992)
3. Broersma, H.J., Kloks, T., Kratsch, D., Müller, H.: Independent Sets in Asteroidal Triple-Free Graphs. SIAM J. Discrete Math. 12, 276–287 (1999)
4. Chudnovsky, M., Seymour, P.D.: The three-in-a-tree problem. Combinatorica 30, 387–417 (2010)

5. Derhy, N., Picouleau, C.: Finding induced trees. Discrete Applied Mathematics 157, 3552–3557 (2009)
6. Corneil, D.G., Olariu, S., Stewart, L.: Asteroidal Triple-Free Graphs. SIAM J. Discrete Math. 10, 299–430 (1997)
7. Corneil, D.G., Olariu, S., Stewart, L.: Linear time algorithms for dominating pairs in asteroidal triple-free graphs. SIAM J. Comput. 28, 1284–1297 (1999)
8. Downey, R.G., Fellows, M.R.: Parameterized complexity. Monographs in Computer Science. Springer, New York (1999)
9. Fellows, M.R.: The Robertson-Seymour theorems: A survey of applications. In: Richter, R.B. (ed.) Proc. AMS-IMS-SIAM Joint Summer Research Conference. Contemporary Mathematics, vol. 89, pp. 1–18. Amer. Math. Soc., Providence (1989)
10. Fiala, J., Kamiński, M., Lidicky, B., Paulusma, D.: The k-in-a-path problem for claw-free graphs. Algorithmica 62, 499–519 (2012)
11. Golovach, P.A., Paulusma, D., van Leeuwen, E.J.: Induced Disjoint Paths in Claw-Free Graphs. arXiv:1202.4419v1 [cs.DM] (2012)
12. Grohe, M., Kawarabayashi, K., Marx, D., Wollan, P.: Finding topological subgraphs is fixed-parameter tractable. In: Proc. STOC, pp. 479–488 (2011)
13. Karp, R.M.: On the complexity of combinatorial problems. Networks 5, 45–68 (1975)
14. Kloks, T., Kratsch, D., Müller, H.: On the structure of graphs with bounded asteroidal number. Graphs and Combinatorics 17, 295–306 (2001)
15. Kobayashi, Y., Kawarabayashi, K.: A linear time algorithm for the induced disjoint paths problem in planar graphs. JCSS 78, 670–680 (2012)
16. Kratsch, D.: Domination and total domination on asteroidal triple-free graphs. Discrete Applied Mathematics 99, 111–123 (2000)
17. Kratsch, D., Müller, H., Todinca, I.: Feedback Vertex Set and Longest Induced Path on AT-Free Graphs. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 309–321. Springer, Heidelberg (2003)
18. Lekkerkerker, C.G., Boland, J.C.: Representation of a finite graph by a set of intervals on the real line. Fund. Math. 51, 45–64
19. Lévêque, B., Lin, D.Y., Maffray, F., Trotignon, N.: Detecting induced subgraphs. Discrete Applied Mathematics 157, 3540–3551 (2009)
20. Lynch, J.F.: The equivalence of theorem proving and the interconnection problem. SIGDA Newsletter 5, 31–36 (1975)
21. McDiarmid, C.J.H., Reed, B.A., Schrijver, A., Shepherd, F.B.: Induced Circuits in Planar Graphs. J. Comb. Theory B 60, 169–176 (1994)
22. Natarajan, S., Sprague, A.P.: Disjoint Paths in Circular Arc Graphs. Nord. J. Comput. 3, 256–270 (1996)
23. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Comb. Theory B 63, 65–110 (1995)
24. Stacho, J.: 3-Colouring AT-Free Graphs in Polynomial Time. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 144–155. Springer, Heidelberg (2010)

Effective Computation of Immersion Obstructions for Unions of Graph Classes*

Archontia C. Giannopoulou, Iosif Salem, and Dimitris Zoros

Department of Mathematics, National and Kapodistrian University of Athens,
Athens, Greece
{arcgian,ysalem,dzoros}@math.uoa.gr

Abstract. In the final paper of the Graph Minors series [*Neil Robertson and Paul D. Seymour. Graph minors XXIII. Nash-Williams' immersion conjecture J. Comb. Theory, Ser. B, 100(2):181–205, 2010.*], N. Robertson and P. Seymour proved that graphs are well-quasi-ordered with respect to the immersion relation. A direct implication of this theorem is that each class of graphs that is closed under taking immersions can be fully characterized by forbidding a finite set of graphs (immersion obstruction set). However, as the proof of the well-quasi-ordering theorem is non-constructive, there is no generic procedure for computing such a set. Moreover, it remains an open issue to identify for which immersion-closed graph classes the computation of those sets can become effective. By adapting the tools that where introduced in [*Isolde Adler, Martin Grohe and Stephan Kreutzer. Computing excluded minors, SODA, 2008: 641-650.*] for the effective computation of obstruction sets for the minor relation, we expand the horizon of the computability of obstruction sets for immersion-closed graph classes. In particular, we prove that there exists an algorithm that, given the immersion obstruction sets of two graph classes that are closed under taking immersions, outputs the immersion obstruction set of their union.

Keywords: Immersions, Obstructions, Unique Linkage Theorem, Tree-width.

1 Introduction

The development of the graph minor theory constitutes a vital part of modern Combinatorics. A lot of theorems proved and techniques introduced in its context, appear to be of crucial importance in Algorithmics and the theory of Parameterized Complexity as well as in Structural Graph Theory. Such examples are the Excluded Grid Theorem [16], the Structure Theorems [15, 17] and

[*] The first author was supported by a grant of the Special Account for Research Grants of the National and Kapodistrian University of Athens (project code: 70/4/10311) and part of her research took place while she was visiting the Department of Applied Mathematics of Charles University in Prague, under the support of the ERC starting grant CCOSA (grant agreement no. 259385).

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 165–176, 2012.

the Irrelevant Vertex Technique [14]. (For examples of algorithmic applications, see [5,10]).

We say that a graph H is an immersion (minor) of a graph G, if we can obtain H from a subgraph of G by lifting (contracting) edges. (For detailed definitions, see Section 2). While the minor relation has been extensively studied throughout the last decades [1,3,14–19], the immersion relation has only recently gained more attention [6,10]. One of the fundamental results that appeared in the last paper of the Graph Minors series was the proof of Nash-Williams' Conjecture, that is, the class of all graphs is well-quasi-ordered by the immersion relation [19].

A direct corollary of these results is that a graph class $\mathcal{C}$, which is closed under taking immersions, can be characterized by a finite family $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ of minimal, according to the immersion relation, graphs that are not contained in $\mathcal{C}$ (called obstructions from now on). Furthermore, in [10], it was proven that there is an $O(|V(G)|^3)$ algorithm that decides whether a graph H is an immersion of a graph G (where the hidden constants depend only on H). Thus, an immediate algorithmic implication of the finiteness of $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ and the algorithm in [10], is that it can be checked in cubic time whether a graph belongs in $\mathcal{C}$ or not (by testing if the graph G contains any of the graphs in $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ as an immersion). In other words, these two results imply that membership in an immersion-closed graph class can be decided in cubic time.

Notice that this meta-algorithmic result assumes that the family $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ is known. However, as the proofs in [17, 19] are non-constructive, there is no generic algorithm that allows us to identify these obstruction sets for every immersion-closed graph class. Moreover, even for fixed graph classes, this task can be extremely challenging as such a set could contain many graphs and no general upper bound on its cardinality is known other than its finiteness. Thus, an open problem is to find out which information on an immersion-closed graph class $\mathcal{C}$ make it is possible to effectively compute the obstruction sets.

The issue of the computability of obstruction sets for minors and immersions was raised by M. Fellows and M. Langston [8,9]. In their papers, they show that the problem of determining obstruction sets from machine descriptions of minor-closed graph classes is recursively unsolvable [9], which directly holds for the immersion relation as well, and propose several methods to tackle the non-constructiveness of these sets. See, for example, [3,8]. Furthermore, the problem of algorithmically identifying minor obstruction sets has been extensively studied [1,3,4,8,9,12].

In this paper, we initiate the study for computing immersion obstruction sets. In particular, we deal with the problem of computing $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ for families of graph classes $\mathcal{C}$ that are constructed by finite unions of immersion-closed graph classes. Notice that the union and the intersection of two immersion-closed graph classes are also immersion-closed, hence their obstruction sets are of finite size. It is also easy to see that, given the obstruction sets of two immersion-closed graph classes, the obstruction set of their intersection can be computed in a trivial way. We prove that there is an algorithm that, given the obstruction sets of two immersion-closed graph classes, outputs the obstruction set of their union.

Our approach is based on the derivation of an upper bound on the tree-width of the obstructions of the graph class. We build on the machinery introduced by I. Adler, M. Grohe and S. Kreutzer in [1] for computing minor obstruction sets. For this, we adapt the results on [1] so to permit the computation of the obstruction set of any immersion-closed graph class, under the conditions that an explicit upper bound on the tree-width of its obstructions can also be computed and the class can be defined in Monadic Second Order Logic. Our next step is a combinatorial result proving an upper bound on the tree-width of the obstructions of the union of two immersion-closed graph classes. We then show that the corresponding obstruction set can be effectively computed. Our combinatorial proofs significantly differ from the ones in [1] and make use of a suitable extension of the Unique Linkage Theorem of K. Kawarabayashi and P. Wollan [11].

The rest of the paper is structured as follows. In Section 2 we state the basic notions that we use throughout the paper as well as few well-known results. In Section 3 we provide a version of Lemma 3.1 of [1], adapted to immersions, in order to prove that the obstruction set of an immersion-closed graph class can be computed when an explicit upper bound of the tree-width of its obstructions is known. In Section 4 we provide the bounds on the tree-width of the graphs that belong in $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_1 \cup \mathcal{C}_2)$ assuming that the sets $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_1)$ and $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_2)$ are known, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are immersion-closed graph classes. The proofs of Lemmata marked with (*) have been omitted due to space constraints.

2 Preliminaries

2.1 Basics

Throughout this paper, graphs are unweighted, undirected and contain no loops or multiple edges. Given a graph G, we denote its set of vertices with $V(G)$, its set of edges with $E(G)$ and the *degree* of a vertex v with $\deg_G(v)$. The *line graph* of a graph G, denoted by $L(G)$, is the graph $(E(G), X)$, where $X = \{\{e_1, e_2\} \subseteq E(G) \mid e_1 \cap e_2 \neq \emptyset \wedge e_1 \neq e_2\}$. Given two graphs G and H, the *lexicographic product* $G \times H$, is the graph with $V(G \times H) = V(G) \times V(H)$ and $E(G \times H) = \{\{(x, y), (x', y')\} \mid (\{x, x'\} \in E(G)) \vee (x = x' \wedge \{y, y'\} \in E(H))\}$.

Given an edge $e = \{x, y\}$ of a graph G, the graph G/e is obtained from G by contracting the edge e, that is, the endpoints x and y are replaced by a new vertex v_{xy} which is adjacent to the old neighbors of x and y (except x and y). A graph H is a *minor* of G, $H \leq_m G$, if there is a function that maps every vertex v of H to a connected set $B_v \subseteq V(G)$, such that for every two distinct vertices v, w of H, B_v and B_w share no common vertex, and for every edge $\{u, v\}$ of H, there is an edge in G with one endpoint in B_v and one in B_u. The graph that is obtained by the union of all B_v such that $v \in V(H)$ and by the edges between B_v and B_u in G, if there exists an edge $\{v, u\}$ in H, is called a *model of H in G*. A model with minimal number of vertices and edges is called *minimal model.*

We say that H is an *immersion* of G (or H is *immersed* in G), $H \leq_{im} G$, if H can be obtained from a subgraph of G after a (possibly empty) sequence of edge lifts, where the *lift* of two edges $e_1 = \{x, y\}$ and $e_2 = \{x, z\}$ to an edge e is the operation of removing e_1 and e_2 from G and then adding the edge $e = \{y, z\}$ in the resulting graph. Equivalently, we say that H is an immersion of G if there is an injective mapping $f : V(H) \to V(G)$ such that, for every edge $\{u, v\}$ of H, there is a path from $f(u)$ to $f(v)$ in G and for any two distinct edges of H the corresponding paths in G are *edge-disjoint*, that is, they do not share common edges. Additionally, if these paths are internally disjoint from $f(V(H))$, then we say that H is *strongly immersed* in G. As above, the function f is called a *model of H in G* and a model with minimal number of vertices and edges is called *minimal model.* A graph class $\mathcal{C}$ is called *immersion-closed*, if for every $G \in \mathcal{C}$ and every H with $H \leq_{im} G$ it holds that $H \in \mathcal{C}$. For example, the class of graphs $\mathcal{E}_t$ that admit a proper edge-coloring of at most t colors such that for every two edges of the same color every path between them contains an edge of greater color is immersion closed. (See [2]). Two paths are called *vertex-disjoint* if they do not share common vertices.

We define an ordering $\leq$ between finite sets of graphs as follows: $\mathcal{F}_1 \leq \mathcal{F}_2$ if and only if

1. $\sum_{G \in \mathcal{F}_1} |V(G)| < \sum_{H \in \mathcal{F}_2} |V(H)|$ or
2. $\sum_{G \in \mathcal{F}_1} |V(G)| = \sum_{H \in \mathcal{F}_2} |V(H)|$ and $\sum_{G \in \mathcal{F}_1} |E(G)| < \sum_{H \in \mathcal{F}_2} |E(H)|$.

Definition 1. *Let $\mathcal{C}$ be an immersion-closed graph class. A set of graphs $F = \{H_1, \ldots, H_n\}$ is called (immersion) obstruction set of $\mathcal{C}$, and is denoted by $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$, if and only if F is a $\leq$-minimal set of graphs for which the following holds: For every graph G, G does not belong to $\mathcal{C}$ if and only if there exists a graph $H \in F$ such that $H \leq_{im} G$.*

Remark 1. We would like to remark here that the obstruction set of an immersion-closed graph class can equivalently be defined in the following way: For any immersion-closed graph class $\mathcal{C}$, the set of its obstructions is the set consisting of all $\leq_{im}$-minimal elements that do not belong in $\mathcal{C}$. However, we also include Definition 1 as it may facilitate the understanding of the intuition behind Lemma 4.

Recall that, because of the seminal result of N. Robertson and P. Seymour [19], for every immersion-closed graph class $\mathcal{C}$, the set $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ is finite.

2.2 Tree-Width and Linkages

A *tree decomposition* of a graph G is a pair (T, B), where T is a tree and B is a function that maps every vertex $v \in V(T)$ to a subset B_v of $V(G)$ such that:

(i) for every edge e of G there exists a vertex t in T such that $e \subseteq B_t$,
(ii) for every $v \in V(G)$, if $r, s \in V(T)$ and $v \in B_r \cap B_s$, then for every vertex t on the unique path between r and s in T, $v \in B_t$ and
(iii) $\cup_{v \in V(T)} B_v = V(G)$.

The width of a tree decomposition (T, B) is $\text{width}(T, B) := \max\{|B_v| - 1 \mid v \in V(T)\}$ and the tree-width of a graph G is the minimum over the $\text{width}(T, B)$, where (T, B) is a tree decomposition of G.

Let r be a positive integer. An *r-approximate linkage* in a graph G is a family L of paths in G such that for every $r + 1$ distinct paths $P_1, P_2, \ldots, P_{r+1}$ in L, it holds that $\bigcap_{i\in[r+1]} V(P_i) = \emptyset$ and each endpoint of the paths appears in exactly one path of L. We call these paths the *components* of the linkage. Let $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ be elements of $V(G)^k$. We say that an r-approximate linkage L, consisting of the paths $P_1, P_2, \ldots, P_k$, *links* $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ if P_i is a path with endpoints α_i and β_i, for every $i \in [k]$. The *order* of such linkage is k. We call an r-approximate linkage of order k, r-approximate k-linkage. Two r-approximate k-linkages L and L' are equivalent if they have the same order and for every component P of L there exists a component P' of L' with the same endpoints. An r-approximate linkage L of a graph G is called *unique* if for every equivalent linkage L' of L, $V(L) = V(L')$. When $r = 1$, such a family of paths is called *linkage*. Finally, a linkage L in a graph G is called *vital* if there is no other linkage in G joining the same pairs of vertices.

In [18], N. Robertson and P. Seymour proved a theorem which is known as The Vital Linkage Theorem. This theorem provides an upper bound for the tree-width of a graph G that contains a vital k-linkage L such that $V(L) = V(G)$, where the bound depends only on k. A stronger statement of the Vital Linkage Theorem was recently proved by K. Kawarabayashi and P. Wollan [11], where instead of asking for the linkage to be vital, it asks for it to be unique. Notice here that a vital linkage is also unique. As in some of our proofs (for example, the proof of Lemma 5) we deal with unique but not necessarily vital linkages we make use of the Vital Linkage Theorem in its latter form which is stated below.

Theorem 1 (The Unique Linkage Theorem [11,18]). *There exists a computable function $w : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let L be a (1-approximate) k-linkage in G with $V(L) = V(G)$. If L is unique then $\mathbf{tw}(G) \leq w(k)$.*

2.3 Monadic Second Order Logic

We now recall some definitions from Monadic Second Order Logic (MSO). An extended introduction to Logic can be found in [7,13].

We call *signature* $\tau = \{R_1, \ldots, R_n\}$ any finite set of relation symbols R_i of any (finite) arity denoted by $\text{ar}(R_i)$. For the language of graphs $\mathcal{G}$ we consider the signature $\tau_{\mathcal{G}} = \{V, E, I\}$ where V represents the set of vertices of a graph G, E the set of edges, and $I = \{(v, e) \mid v \in e \text{ and } e \in E(G)\}$ the incidence relation.

A τ-structure $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}})$ consists of a finite universe A, and the interpretation of the relation symbols R_i of τ in A, that is, for every i, $R_i^{\mathfrak{A}}$ is a subset of $A^{\text{ar}(R_i)}$.

In MSO formulas are defined inductively from atomic formulas, that is, expressions of the form $R_i(x_1, x_2, \ldots, x_{\text{ar}(R_i)})$ or of the form $x = y$ where x_i,

$i \in [\mathrm{ar}(R_i)]$, x and y are variables, by using the Boolean connectives $\neg, \wedge, \vee, \rightarrow$, and existential or universal quantification over individual variables and sets of variables. Furthermore, quantification takes place over vertex or edge variables or vertex-set or edge-set variables. Notice that in the language of graphs the atomic formulas are of the form $V(u), E(e)$ and $I(u, e)$, where u and e are vertex and edge variables respectively.

A *graph structure* $\mathfrak{G} = (V(G) \cup E(G), V^{\mathfrak{G}}, E^{\mathfrak{G}}, I^{\mathfrak{G}})$ is a $\tau_{\mathcal{G}}$-structure, which represents the graph $G = (V, E)$. From now on, we abuse notation by treating G and $\mathfrak{G}$ equally.

A graph class $\mathcal{C}$ is *MSO-definable* if there exists an MSO formula $\phi_{\mathcal{C}}$ in the language of graphs such that $G \in \mathcal{C}$ if and only if $G \models \phi_{\mathcal{C}}$, that is, $\phi_{\mathcal{C}}$ is true in the graph G (G models $\phi_{\mathcal{C}}$).

Lemma 1. *The class of graphs that contain a fixed graph H as an immersion is MSO-definable by an MSO-formula ϕ_H.*

Proof. Let $V(H) = \{v_1, v_2, \ldots, v_n\}$ and $E(H) = \{e_1, e_2, \ldots, e_m\}$. Let also ϕ_H be the following formula.

$$\begin{aligned}\phi_H := \exists E_1, E_2, \ldots, E_m \exists x_1, x_2, \ldots, x_n \Big[& (\bigwedge_{i \in [n]} V(x_i)) \wedge (\bigwedge_{j \in [m]} E_i \subseteq E) \wedge \\ & (\bigwedge_{i \neq j} x_i \neq x_j) \wedge (\bigwedge_{p \neq q} E_p \cap E_q = \emptyset) \wedge \\ & (\bigwedge_{e_r = \{v_k, v_l\} \in E(H)} \mathrm{path}(x_k, x_l, E_r)) \Big],\end{aligned}$$

where $\mathrm{path}(x, y, Z)$ is the MSO formula stating that the edges in Z form a path from x to y. This can be done by saying that the set Z of edges is connected and every vertex v incident to an edge in Z is either incident to exactly two edges of Z or to exactly one edge with further condition that $v = x$ or $v = z$.

Theorem 2 (Seese's Theorem [20]). *For every positive integer k, it is decidable given an MSO-formula whether it is satisfied by a graph G whose tree-width is upper bounded by k.*

In [1], I. Adler, M. Grohe and S. Kreutzer provide tools that allow us to use Seese's theorem, when an upper bound on the tree-width of the obstructions is known and an MSO-description of the graph class can be computed, in order to compute the obstruction sets of minor-closed graph classes. We adapt their machinery to the immersion relation and prove that the tree-width of the obstructions of immersion-closed graph classes is upper bounded by some function that only depends on the graph class. This provides a generic technique to construct immersion obstruction sets when the explicit value of the function is known. Then, by obtaining such an explicit upper bound on the tree-width of the graphs in $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$, where $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ and $\mathcal{C}_1$, $\mathcal{C}_2$ are immersion-closed graph classes whose obstruction sets are given, we show that the set $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ can be effectively computed.

3 Computing Immersion Obstruction Sets

In this Section we state the analog of Lemma 2.2 in [1] (Lemma 2) and the analog of Lemma 3.1 in [1] (Lemma 4) for the immersion relation. Their proofs are omitted due to space constraints. However, we include the necessary definitions and intermediate lemmata in order to give a rough sketch of the framework needed in order to prove our main result.

We first state the combinatorial Lemma of this Section.

Lemma 2. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let H and G be graphs such that $H \leq_{im} G$. If G' is a minimal subgraph of G with $H \leq_{im} G'$ then $\mathbf{tw}(G') \leq f(|E(H)|)$.*

The proof of Lemma 2 is omitted as a stronger statement will be proved later on (Lemma 7). We continue by giving the necessary definitions in order to state the analog of Lemma 3.1 in [1] for the immersion relation.

Extension of MSO. For convenience, we consider the extension of the signature $\tau_{\mathcal{G}}$ to a signature τ_{ex} that pairs the representation of a graph G with the representation of one of its tree-decompositions.

Definition 2. *If G is a graph and $\mathcal{T} = (T, B)$ is a tree-decomposition of G, τ_{ex} is the signature that consists of the relation symbols V, E, I of $\tau_{\mathcal{G}}$, and four more relation symbols V_T, E_T, I_T and B.*
A tree-dec expansion *of G and $\mathcal{T}$, is a τ_{ex}-structure*

$$\mathfrak{G}_{ex} = (V(G) \cup E(G) \cup V(T) \cup E(T), \\ V^{\mathfrak{G}_{ex}}, E^{\mathfrak{G}_{ex}}, I^{\mathfrak{G}_{ex}}, V_T^{\mathfrak{G}_{ex}}, E_T^{\mathfrak{G}_{ex}}, I_T^{\mathfrak{G}_{ex}}, B^{\mathfrak{G}_{ex}})$$

where $V_T^{\mathfrak{G}_{ex}} = V(T)$ represents the node set of T, $E_T^{\mathfrak{G}_{ex}} = E(T)$ the edge set of T, $I_T^{\mathfrak{G}_{ex}} = \{(v, e) \mid v \in e \cap V(T) \wedge e \in E(T)\}$ the incidence relation in T and $B^{\mathfrak{G}_{ex}} = \{(t, v) \mid t \in V(T) \wedge v \in B_t \cap V(G)\}$.

We denote by $\mathcal{C}_{\mathcal{T}_k}$ the class of tree-dec expansions consisting of a graph G with $\mathbf{tw}(G) \leq k$, and a tree decomposition (T, B) of G of width$(T, B) \leq k$.

Lemma 3 ([1])

1. *Let G be a graph and (T, B) a tree decomposition of it with width$(T, B) \leq k$. Then, the tree-width of the tree-dec expansion of G is at most $k + 2$.*
2. *There is an MSO-sentence $\phi_{\mathcal{C}_{\mathcal{T}_k}}$ such that for every τ_{ex}-structure $\mathfrak{G}$, $\mathfrak{G} \models \phi_{\mathcal{C}_{\mathcal{T}_k}}$ if and only if $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_k}$.*

A classic result of Seese [20] (see Theorem 2) states that we can decide, for every $k \geq 0$, if an MSO-formula is satisfied in a graph G of $\mathbf{tw}(G) \leq k$. An immediate corollary of this result and Lemma 3 is the following.

Corollary 1. *We can decide, for every k, if an MSO-formula ϕ is satisfied in some $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_k}$.*

Theorem 3 ([1]). *For every $k \geq 0$, there is an MSO-sentence $\phi_{\mathcal{T}_k}$ such that for every tree-dec expansion $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_l}$ of G, for some $l \geq k$, it holds that $\mathfrak{G} \models \phi_{\mathcal{T}_k}$ if and only if $\mathbf{tw}(G) = k$.*

Definition 3. *A graph class $\mathcal{C}$ is* layer-wise MSO-definable*, if for every $k \in \mathbb{N}$ we can compute an MSO-formula ϕ_k such that $G \in \mathcal{C} \wedge \mathbf{tw}(G) \leq k$ if and only if $\mathfrak{G} \models \phi_k$, where $\mathfrak{G} \in \mathcal{C}_{\mathcal{T}_k}$ is the tree-dec expansion of G.*

Definition 4. *Let $\mathcal{C}$ be an immersion-closed graph class. The* width *of $\mathcal{C}$, $\mathbf{width}(\mathcal{C})$ is the minimum positive integer k such that for every graph $G \notin \mathcal{C}$ there is a graph $G' \subseteq G$ with $G' \notin \mathcal{C}$ and $\mathbf{tw}(G') \leq k$.*

Note that Lemma 2 ensures that the width of an immersion-closed graph class is well-defined.

Observation 1. *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are immersion-closed graph classes then the following hold.*

1. *For every graph $G \notin \mathcal{C}_1 \cup \mathcal{C}_2$, there exists a graph $G' \subseteq G$ such that $G' \notin \mathcal{C}_1 \cup \mathcal{C}_2$ and $\mathbf{tw}(G') \leq \max\{r(|E(H)|, |E(J)|) \mid H \in \mathbf{obs}_{\leq_{im}}(\mathcal{C}_1), J \in \mathbf{obs}_{\leq_{im}}(\mathcal{C}_2)\}$, where r is the function of Lemma 7 and thus,*
2. *For every graph $G \notin \mathcal{C}_1$, there exists a graph $G' \subseteq G$ such that $G' \notin \mathcal{C}_1$ and $\mathbf{tw}(G') \leq \max\{f(|E(H)|) \mid H \in \mathbf{obs}_{\leq_{im}}(\mathcal{C}_1)\}$, where f is the function of Lemma 2.*

Finally, we state the analog of Lemma 3.1 in [1] for the immersion relation.

Lemma 4 (*). *There exists an algorithm that, given an upper bound $l \geq 0$ on the width of a layer-wise MSO-definable class $\mathcal{C}$, and a computable function $f : \mathbb{N} \to MSO$ such that for every positive integer k, $f(k) = \phi_k$, where ϕ_k is the MSO-formula defining $\mathcal{C} \cap \mathcal{T}_k$, it computes $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$.*

Remark 2. We remark here that Lemma 4 provides a generic algorithm for computing the obstruction set of any immersion-closed graph class, given that the conditions stated are satisfied. However, notice that the above lemma implies that there is an algorithm that given an MSO formula ϕ and $k \in \mathbb{N}$, so that ϕ defines an immersion closed-graph class $\mathcal{C}$ of width at most k, computes the obstruction set of $\mathcal{C}$.

The only missing ingredient towards our ultimate goal is an explicit upper bound on the tree-width of the obstructions of the graph class $\mathcal{C}_1 \cup \mathcal{C}_2$ where $\mathcal{C}_1$ and $\mathcal{C}_2$ are immersion-closed graph classes. In the following Section we prove that such a bound can be computed, given the obstruction sets of $\mathcal{C}_1$ and $\mathcal{C}_2$.

4 Tree-Width Bounds for the Obstructions

We first prove the following generalization of the Unique Linkage Theorem.

Lemma 5. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. Let G be a graph that contains a 2-approximate k-linkage $\tilde{L}$ such that $V(\tilde{L}) = V(G)$. If $\tilde{L}$ is unique, then* $\mathbf{tw}(G) \leq f(k)$.

Proof. Let G be a graph that contains a unique 2-approximate k-linkage $\tilde{L}$ with $V(\tilde{L}) = V(G)$ that links $A = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $B = (\beta_1, \beta_2, \ldots, \beta_k)$ in G. Denote by T the set $A \cup B$ and consider the graph G^b with

$$\begin{aligned} V(G^b) &= V((G \setminus T) \times K_2) \cup T \\ E(G^b) &= E((G \setminus T) \times K_2) \cup \{\{t, t'\} \mid t, t' \in T \wedge \{t, t'\} \in E(G)\} \\ &\quad \cup \{\{t, (v, x)\} \mid t \in T \wedge x \in V(K_2) \wedge v \in V(G) \wedge \{t, v\} \in E(G)\}, \end{aligned}$$

where $V(K_2) = \{1, 2\}$. It is easy to see that G^b contains a k-linkage that links A and B. Let G' be a minimal induced subgraph of G^b that contains a k-linkage L' that links A and B. From Theorem 1, it follows that

$$\mathbf{tw}(G') \leq w(k). \tag{1}$$

From now on we work towards proving that $G \leq_m G'$. In order to achieve this, we prove the following two claims for G'.

Claim. If L' is a k-linkage in G' that links A and B then for every vertex $v \in V(G) \setminus T$ no path of L' contains both $(v, 1)$ and $(v, 2)$.

Proof. Towards a contradiction, assume that for some vertex $v \in V(G) \setminus T$, there exists a (t, t')-path P of L' that contains both $(v, 1)$ and $(v, 2)$. Without loss of generality, assume also that $(v, 1)$ appears before $(v, 2)$ in P. Let y be the successor of $(v, 2)$ in P and notice that $y \neq (v, 1)$. From the definition of G^b and the fact that G' is an induced subgraph of G^b, $\{y, (v, 1)\} \in E(G') \setminus E(L')$. By replacing the subpath of P from $(v, 1)$ to y with the edge $\{(v, 1), y\}$, we obtain a linkage in $G' \setminus (v, 2)$ that links A and B. This contradicts to the minimality of G'. □

Claim. If L' is a k-linkage in G' that links A and B then for every vertex $v \in V(G) \setminus T$, $V(L') \cap \{(v, 1), (v, 2)\} \neq \emptyset$.

Proof. Assume, in contrary, that there exists a linkage L' in G' and a vertex $x \in V(G) \setminus T$ such that L' links A and B and $V(L') \cap \{(x, 1), (x, 2)\} = \emptyset$. Claim 4 ensures that, after contracting the edges $\{(v, 1), (v, 2)\}$, $v \in V(G) \setminus T$ (whenever they exist), the corresponding paths compose a 2-approximate k-linkage $\tilde{L}'$ of $G \setminus \{x\}$ that links A and B. This is a contradiction to the assumption that $\tilde{L}$ is unique. Thus, the claim holds. □

Recall that $T \subseteq V(G')$ and that G' is an induced subgraph of G^b. Claim 4 implies that we may obtain G from G' by contracting the edges $\{(v, 1), (v, 2)\}$ for every $v \in V(G) \setminus T$ (whenever they exist). As $G \leq_m G'$, from (1), it follows that, $\mathbf{tw}(G) \leq w(k)$. □

We remark that, the previous lemma holds for any graph G that contains an r-approximate k-linkage. This can be seen by substituting $(G \setminus T) \times K_2$ with $(G \setminus T) \times K_r$ in its proof.

We now state a lemma that provides the upper bound of a graph G, given the upper bound of its linear graph $L(G)$.

Lemma 6 (*). *If G is a graph and k is a positive integer with $\mathbf{tw}(L(G)) \leq k$ then $\mathbf{tw}(G) \leq 2k+1$.*

Before we proceed to the next lemma, we need to introduce the notion of an r-approximate k-edge-linkage in a graph. Similarly to the notion of an r-approximate linkage, an *r-approximate edge-linkage* in a graph G is a family of paths E in G such that for every $r+1$ distinct paths $P_1, P_2, \ldots, P_{r+1}$ in E, it holds that $\cap_{i \in [r+1]} E(P_i) = \emptyset$. We call these paths the *components* of the edge-linkage. Let $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ be elements of $V(G)^k$. We say that an r-approximate edge-linkage E, consisting of the paths $P_1, P_2, \ldots, P_k$, *links* $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and $(\beta_1, \beta_2, \ldots, \beta_k)$ if P_i is a path with endpoints α_i and β_i, for every $i \in [k]$. The *order* of E is k. We call an r-approximate edge-linkage of order k, r-approximate k-edge-linkage. When $r = 1$, we call such a family of paths, an *edge-linkage.*

Lemma 7. *There exists a computable function r such that the following holds. Let G_1, G_2 and G be graphs such that $G_i \leq_{im} G$, $i = 1, 2$. If G' is a minimal subgraph of G where $G_i \leq_{im} G'$, $i = 1, 2$, then $\mathbf{tw}(G') \leq r(|E(G_1)|, |E(G_2)|)$.*

Proof. Let G' be a minimal subgraph of G such that $G_i \leq_{im} G'$, $i = 1, 2$. Recall that the edges of G_i compose a k_i-edge-linkage E_i in G, where $k_i = |E(G_i)|$, $i = 1, 2$. Furthermore, observe that the paths of E_1 and E_2 constitute a 2-approximate k-edge-linkage E of G, where $k = k_1 + k_2$. Indeed, notice that in contrary to linkages, we do not require the paths that are forming edge-linkages to have different endpoints. The minimality of G' implies that $\bigcup\{P \mid P \in E\} = G'$. Denote by $A = (v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ and $B = (v_{j_1}, v_{j_2}, \ldots, v_{j_k})$ the vertex sets that are edge-linked by E in G' and let $\widehat{G}$ be the graph with

$$V(\widehat{G}) = V(G') \cup \{u_{i_q} \mid q \in [k]\} \cup \{u_{j_q} \mid q \in [k]\},$$
$$E(\widehat{G}) = E(G') \cup \{t_{i_q} \mid q \in [k]\} \cup \{t_{j_q} \mid q \in [k]\},$$

where the vertices u_{i_q} and u_{j_q}, $q \in [k]$ are new, $t_{i_q} = \{u_{i_q}, v_{i_q}\}$, $q \in [k]$ and $t_{j_q} = \{u_{j_q}, v_{j_q}\}$, $q \in [k]$.

Consider the line graph of $\widehat{G}$, $L(\widehat{G})$, and notice that E corresponds to a 2-approximate k-linkage L from A_L to B_L in $L(\widehat{G})$, where $A_L = (t_{i_1}, t_{i_2}, \ldots, t_{i_k})$ and $B_L = (t_{j_1}, t_{j_2}, \ldots, t_{j_k})$. This is true as, from the construction of $\widehat{G}$, all the vertices in A_L and B_L are distinct. The minimality of G' yields that $V(L) = V(L(\widehat{G}))$ and implies that L is unique. From Lemma 5, we obtain that $\mathbf{tw}(L(\widehat{G})) \leq f(k)$. Therefore, from Lemma 6, we get that $\mathbf{tw}(\widehat{G}) \leq p(f(k))$, where p is the function of Lemma 6. Finally, as $G' \subseteq \widehat{G}$, $\mathbf{tw}(G') \leq r(k_1, k_2)$, where $r(k_1, k_2) = p(f(k_1 + k_2))$. □

Notice that Lemma 2 follows from Lemma 7 when we set G_2 to be the empty graph. Finally, we show that given two immersion closed graph classes $\mathcal{C}_1$ and $\mathcal{C}_2$ the immersion-closed graph class $\mathcal{C}_1 \cup \mathcal{C}_2$ is layer-wise MSO-definable.

Observation 2. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be immersion-closed graph classes, then $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ is a layer-wise MSO-definable class defined, for every $k \geq 0$, by the formula*

$$\phi_k \equiv \left(\left(\bigwedge_{G \in \mathbf{obs}_{\leq_{im}}(\mathcal{C}_1)} \neg\phi_G \right) \vee \left(\bigwedge_{H \in \mathbf{obs}_{\leq_{im}}(\mathcal{C}_2)} \neg\phi_H \right) \right) \wedge \phi_{\mathcal{T}_k}$$

where ϕ_G and ϕ_H are the formulas described in Lemma 1, and $\phi_{\mathcal{T}_k}$ the formula of Theorem 3 .

We are now able to prove the Main Theorem.

Theorem 4. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two immersion-closed graph classes. If the sets $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_1)$ and $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_2)$ are given, then the set $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_1 \cup \mathcal{C}_2)$ is computable.*

Proof. According to Observation 2, $\mathcal{C}_1 \cup \mathcal{C}_2$ is a layer-wise MSO-definable class, and according to Lemma 7 there is a bound on the width of $\mathcal{C}_1 \cup \mathcal{C}_2$. Therefore, Lemma 4 is applicable. □

5 Conclusions and Further Work

In this paper, we further the study on the constructibility of obstruction sets for immersion-closed graph classes. In particular, we provide an explicit upper bound on the obstructions of a graph class $\mathcal{C}$, which is the union of two immersion-closed graph classes $\mathcal{C}_1$ and $\mathcal{C}_2$ with $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_1)$ and $\mathbf{obs}_{\leq_{im}}(\mathcal{C}_2)$ given. Then, using that result, we prove that $\mathbf{obs}_{\leq_{im}}(\mathcal{C})$ is computable.

In [19], N. Robertson and P. Seymour claimed that the class of graphs is also well-quasi-ordered under the strong immersion relation. However, a full proof of this result has not appeared so far. We remark that the combinatorial results of this paper, that is, the explicit upper bounds on the tree-width of the obstructions, also hold for the strong immersion relation. Thus, if the claim of N. Robertson and P. Seymour holds, the obstruction set of the union of two strongly immersion-closed graph classes, whose obstruction sets are given, can be effectively computed.

Finally, it was proven by B. Courcelle, R. Downey and M. Fellows [4] that the obstruction set of a minor-closed graph class $\mathcal{C}$ cannot be computed by an algorithm whose input is a description of $\mathcal{C}$ as an MSO-sentence. The computability of the obstruction set of an immersion-closed graph class $\mathcal{C}$, given an MSO description of $\mathcal{C}$, remains an open problem.

Acknowledgements. We would like to thank Prof. Dimitrios M. Thilikos for suggesting this problem to us. We also wish to thank the anonymous referees for helpful comments and remarks that improved the presentation of our results.

References

1. Adler, I., Grohe, M., Kreutzer, S.: Computing excluded minors. In: Teng, S.-H. (ed.) SODA, pp. 641–650. SIAM (2008)
2. Bodlaender, H.L., Deogun, J.S., Jansen, K., Kloks, T., Kratsch, D., Müller, H., Tuza, Z.: Rankings of graphs. SIAM J. Discrete Math. 11(1), 168–181, (electronic) (1998)
3. Cattell, K., Dinneen, M.J., Downey, R.G., Fellows, M.R., Langston, M.A.: On computing graph minor obstruction sets. Theor. Comput. Sci. 233(1-2), 107–127 (2000)
4. Courcelle, B., Downey, R.G., Fellows, M.R.: A note on the computability of graph minor obstruction sets for monadic second order ideals. J. UCS 3(11), 1194–1198 (1997)
5. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. Journal of the ACM 52(6), 866–893 (2005)
6. DeVos, M., Dvořák, Z., Fox, J., McDonald, J., Mohar, B., Scheide, D.: Minimum degree condition forcing complete graph immersion. ArXiv e-prints (January 2011)
7. Enderton, H.B.: A mathematical introduction to logic. Academic Press (1972)
8. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. J. Assoc. Comput. Mach. 35(3), 727–739 (1988)
9. Fellows, M.R., Langston, M.A.: On search, decision, and the efficiency of polynomial-time algorithms. J. Comput. System Sci. 49(3), 769–779 (1994)
10. Grohe, M., Kawarabayashi, K.I., Marx, D., Wollan, P.: Finding topological subgraphs is fixed-parameter tractable. In: STOC, pp. 479–488 (2011)
11. Kawarabayashi, K.I., Wollan, P.: A shorter proof of the graph minor algorithm: the unique linkage theorem. In: Schulman, L.J. (ed.) STOC, pp. 687–694. ACM (2010)
12. Lagergren, J.: The Size of an Interwine. In: Abiteboul, S., Shamir, E. (eds.) ICALP 1994. LNCS, vol. 820, pp. 520–531. Springer, Heidelberg (1994)
13. Mendelson, E.: Introduction to mathematical logic, 3rd edn. Chapman and Hall (1987)
14. Robertson, N., Seymour, P.D.: Graph minors. XXII. Irrelevant vertices in linkage problems (to appear)
15. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. Journal of Combinatorial Theory. Series B 63(1), 65–110 (1995)
16. Robertson, N., Seymour, P., Thomas, R.: Quickly excluding a planar graph. J. Combin. Theory Ser. B 62(2), 323–348 (1994)
17. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. J. Comb. Theory, Ser. B 89(1), 43–76 (2003)
18. Robertson, N., Seymour, P.D.: Graph minors. XXI. Graphs with unique linkages. J. Comb. Theory, Ser. B 99(3), 583–616 (2009)
19. Robertson, N., Seymour, P.D.: Graph minors XXIII. Nash-Williams' immersion conjecture. J. Comb. Theory, Ser. B 100(2), 181–205 (2010)
20. Seese, D.: The structure of models of decidable monadic theories of graphs. Ann. Pure Appl. Logic 53(2), 169–195 (1991)

Algorithms on Minimizing the Maximum Sensor Movement for Barrier Coverage of a Linear Domain*

Danny Z. Chen[1], Yan Gu[2], Jian Li[3], and Haitao Wang[1,**]

[1] Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46556, USA
{dchen,hwang6}@nd.edu

[2] Department of Computer Science and Technology
Tsinghua University, Beijing 100084, China
henryy321@gmail.com

[3] Institute for Interdisciplinary Information Sciences (IIIS)
Tsinghua University, Beijing 100084, China
lijian83@mail.tsinghua.edu.cn

Abstract. In this paper, we study the problem of moving n sensors on a line to form a barrier coverage of a specified segment of the line such that the maximum moving distance of the sensors is minimized. Previously, it was an open question whether this problem on sensors with arbitrary sensing ranges is solvable in polynomial time. We settle this open question positively by giving an $O(n^2 \log n \log\log n)$ time algorithm. Further, if all sensors have the same-size sensing range, we give an $O(n \log n)$ time algorithm, which improves the previous best $O(n^2)$ time solution.

1 Introduction

A Wireless Sensor Network (WSN) uses a large number of sensors to monitor some surrounding environmental phenomena [1]. Intrusion detection and border surveillance constitute a major application category for WSNs. A main goal of these applications is to detect intruders as they cross the boundary of a region or domain. For example, research efforts were made to extend the scalability of WSNs to the monitoring of international borders [10,11]. Unlike the traditional *full coverage* [13,17,18] which requires an entire target region to be covered by the sensors, the *barrier coverage* [2,3,7,8,11] only seeks to cover the perimeter of the region to ensure that any intruders are detected as they cross the region border. Since barrier coverage requires fewer sensors, it is often preferable to

* Chen and Wang's research was supported in part by NSF under Grant CCF-0916606. Li's research was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

** Corresponding author. This work was partially done while the author was visiting at IIIS at Tsinghua University.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 177–188, 2012.

full coverage. Because sensors have limited battery-supplied energy, it is desired to minimize their movements. In this paper, we study a linear barrier coverage problem where the barrier is for a (finite) line segment and the sensors are initially located on the line containing the barrier segment and allowed to move on the line. As discussed in the previous work [7,8,15] and shown in this paper, barrier coverage even for linear domains poses some challenging algorithmic issues. Also, our solutions may be used in solving more general problems. For example, if the barrier is sought for a simple polygon, then we may consider each of its edges separately and apply our algorithms to each edge.

In our problem, each sensor has a *sensing range* (or *range* for short) and we want to move the sensors to form a coverage for the barrier such that the maximum sensor movement is minimized.

1.1 Problem Definitions, Previous Work, and Our Results

Denote by $B = [0, L]$ the barrier that is a line segment from $x = 0$ to $x = L > 0$ on the x-axis. A set $S = \{s_1, s_2, \ldots, s_n\}$ of n mobile sensors are initially on the x-axis. Each sensor $s_i \in S$ has a range $r_i > 0$ and is located at the coordinate x_i. We assume $x_1 \leq x_2 \leq \cdots \leq x_n$. If a sensor s_i is at the position x', then we say s_i *covers* the interval $[x' - r_i, x' + r_i]$, called the *covering interval* of s_i. Our problem is to find a set of *destinations* on the x-axis, $\{y_1, y_2, \ldots, y_n\}$, for the sensors (i.e., for each $s_i \in S$, move s_i from x_i to y_i) such that each point on the barrier B is covered by at least one sensor and the maximum moving distance of the sensors (i.e., $\max_{1 \leq i \leq n}\{|x_i - y_i|\}$) is minimized. We call this problem the *barrier coverage on a line segment*, denoted by BCLS. We assume $2 \cdot \sum_{i=1}^{n} r_i \geq L$ (otherwise, a barrier coverage for B is not possible).

The *decision version* of BCLS is defined as follows. Given a value $\lambda \geq 0$, determine whether there is a *feasible solution* in which the moving distance of each sensor is at most λ. If the ranges of all sensors are the same (i.e., the r_i's are all equal), then we call it the *uniform case* of BCLS. When the sensors have arbitrary ranges, we call it the *general case*.

The BCLS problem has been studied before. The uniform case has been solved in $O(n^2)$ time [7]. An $O(n)$ time algorithm is also given in [7] for the decision version of the uniform case. However, it has been open whether the general case is solvable in polynomial time [7].

In this paper, we settle the open problem on the general BCLS by presenting an $O(n^2 \log n \log\log n)$ time algorithm. We also solve the decision version of the general BCLS in $O(n \log n)$ time. For the uniform case, we derive an $O(n \log n)$ time algorithm, improving the previous $O(n^2)$ time solution [7]; and further, if all sensors are initially on B, our algorithm runs in $O(n)$ time.

1.2 Related Work

A variation of the decision version of the general BCLS is shown to be NP-hard [7]. Additional results were also given in [7] for the case $2 \cdot \sum_{i=1}^{n} r_i < L$.

Mehrandish *et al.* [15] also considered the line segment barrier, but unlike the BCLS problem, they intended to use the minimum number of sensors to form a barrier coverage, which they proved to be NP-hard. But, if all sensors have the same range, polynomial time algorithms were possible [15]. Another study of the line segment barrier [8] aimed to minimize the sum of the moving distances of all sensors; this problem is NP-hard [8], but is solvable in polynomial time when all sensors have the same range [8]. In addition, Li *et al.* [12] considers the linear coverage problem which aims to set an energy for each sensor to form a coverage such that the cost of all sensors is minimized. There [12], the sensors are not allowed to move, and the more energy a sensor has, the larger the covering range of the sensor and the larger the cost of the sensor.

Bhattacharya *et al.* [2] studied a 2-D barrier coverage problem in which the barrier is a circle and the sensors, initially located inside the circle, are moved to the circle to form a coverage such that the maximum sensor movement is minimized; the ranges of the sensors are not explicitly specified but the destinations of the sensors are required to form a regular n-gon on the circle. Subsequent improvements of the results in [2] have been made [4,16].

Some other barrier coverage problems have been studied. For example, Kumar *et al.* [11] proposed algorithms for determining whether a region is barrier covered after the sensors are deployed. They considered both the deterministic version (the sensors are deployed deterministically) and the randomized version (the sensors are deployed randomly), and aimed to determine a barrier coverage with high probability. Chen *et al.* [3] introduced a local barrier coverage problem in which individual sensors determine the barrier coverage locally.

2 An Overview of Our Approaches

Throughout the paper, for any problem we consider, let λ^* denote the maximum sensor movement in an optimal solution.

For the uniform BCLS, as shown in [7], a key property is that there always exists an *order preserving* optimal solution, i.e., the order of the sensors in the optimal solution is the same as that in the input. Based on this property, the previous $O(n^2)$ time algorithm [7] covers B from left to right; in each step, it picks the next sensor and re-balances the current maximum sensor movement. We take a very different approach. With the order preserving property, we determine a set Λ of candidate values for λ^* with $\lambda^* \in \Lambda$. Consequently, by using the decision algorithm, we can find λ^* in Λ. But, this approach may be inefficient since $|\Lambda| = \Theta(n^2)$. To reduce the running time, our strategy is not to compute the set Λ explicitly. Instead, we compute an element in Λ whenever we need it. A possible attempt would be to first find a sorted order for the elements of Λ or (implicitly) sort the elements of Λ, and then obtain λ^* by binary search. However, it seems not easy to (implicitly) sort the elements of Λ. Instead, based on several new observations, we manage to find a way to partition the elements of Λ into n sorted lists, each list containing $O(n)$ elements. Next, by using a technique called *binary search on sorted arrays* [5], we are able to find λ^* in Λ

in $O(n \log n)$ time. For the special case when all sensors are initially located on B, a key observation we make is that λ^* is precisely the maximum value of the candidate set Λ. Although $\Lambda = \Theta(n^2)$, based on new observations, its maximum value can be computed in $O(n)$ time. Due to the space limit, our algorithms for the uniform BCLS are omitted and can be found in the full version of this paper.

For the general BCLS, as indicated in [7], the order preserving property no longer holds. Consequently, our approach for the uniform case does not work. The main difficulty of this case is that we do not know the order of the sensors appeared in an optimal solution. Due to this difficulty, no polynomial time algorithm was known before for the general BCLS. To solve this problem, we first develop a greedy algorithm for the decision version of the general BCLS. After $O(n \log n)$ time preprocessing, our decision algorithm takes $O(n \log \log n)$ time for any value λ. If $\lambda \geq \lambda^*$, implying that there exists a feasible solution, then our decision algorithm can determine the order of sensors in a feasible solution for covering B. For the general BCLS, we seek to simulate the behavior of the decision algorithm on $\lambda = \lambda^*$. Although we do not know the value λ^*, our algorithm determines the same sensor order as it would be obtained by the decision algorithm on the value $\lambda = \lambda^*$. To this end, each step of the algorithm uses our decision algorithm as a decision procedure. The idea is somewhat similar to parametric search [6,14], and here we "parameterize" our decision algorithm. However, unlike the typical parametric search [6,14], our approach does not involve any parallel scheme and is practical.

For ease of exposition, we assume that initially no two sensors are located at the same position (i.e., $x_i \neq x_j$ for any $i \neq j$), and the covering intervals of any two different sensors do not share a common endpoint. Our algorithms can be easily generalized to the general situation.

In the following, we discuss the decision version of the general BCLS in Section 3. In Section 4, we present our algorithm for the general BCLS, which we refer to as the *optimization version* of the problem. Due to the space limit, some proofs are omitted and can be found in the full version of this paper.

For each sensor $s_i \in S$, we call the right (resp., left) endpoint of the covering interval of s_i the *right* (resp., *left*) *extension* of s_i. Each of the right and left extensions of s_i is an *extension* of s_i. Denote by $p(x')$ the point on the x-axis whose coordinate is x', and denote by $p^+(x')$ (resp., $p^-(x')$) a point to the right (resp., left) of $p(x')$ and infinitely close to $p(x')$. The concept of $p^+(x')$ and $p^-(x')$ is only used to explain the algorithms, and we never need to find such a point. Note that we can easily determine whether $\lambda^* = 0$, say, in $O(n \log n)$ time. Henceforth, we assume $\lambda^* > 0$.

3 The Decision Version of the General BCLS

Given any value λ, the decision version is to determine whether $\lambda^* \leq \lambda$. Below, we first explore some properties of a feasible solution for λ.

By a sensor *configuration*, we refer to a specification of where each sensor $s_i \in S$ is located. By this definition, the input is a configuration in which each sensor s_i is located at x_i. The *displacement* of a sensor in a configuration C is the distance between the position of the sensor in C and its original position in the input. A configuration C is a *feasible solution* for the distance λ if the sensors in C form a barrier coverage of B (i.e., the union of the covering intervals of the sensors in C contains B) and the displacement of each sensor is at most λ. In a feasible solution, a subset $S' \subseteq S$ is called a *solution set* if the sensors in S' form a barrier coverage; of course, S itself is also a solution set. A feasible solution may have multiple solution sets. A sensor s_i in a solution set S' is said to be *critical* with respect to S' if s_i covers a point on B that is not covered by any other sensor in S'. If every sensor in S' is critical, then S' is called a *critical set.*

Given any value λ, if $\lambda \geq \lambda^*$, then our decision algorithm will find a critical set and determine the order in which the sensors of the critical set will appear in a feasible solution for λ. Consider a critical set S^c. For each sensor $s \in S^c$, we call the set of points on B that are covered by s but not covered by any other sensor in S^c the *exclusive coverage* of s. The proof of Observation 1 is omitted.

Observation 1. *The exclusive coverage of each sensor in a critical set S^c is a continuous portion of the barrier B.*

For a critical set S^c in a feasible solution SOL, we define the *cover order* of the sensors in S^c as the order of these sensors in SOL such that their exclusive coverages are from left to right.

Observation 2. *The cover order of the sensors of a critical set S^c in a feasible solution SOL is consistent with the left-to-right order of the positions of these sensors in SOL. Further, the cover order is also consistent with the order of the* right *(resp., left) extensions of these sensors in SOL.*

Proof. Consider any two sensors s_i and s_j in S^c with ranges r_i and r_j, respectively. Without loss of generality, assume s_i is to the left of s_j in the cover order, i.e., the exclusive coverage of s_i is to the left of that of s_j in SOL. Let y_i and y_j be the positions of s_i and s_j in SOL, respectively. To prove the observation, it suffices to show $y_i < y_j$, $y_i + r_i < y_j + r_j$, and $y_i - r_i < y_j - r_j$.

Let p be a point in the exclusive coverage of s_j. We also use p to denote its coordinate on the x-axis. Then p is not covered by s_i, implying either $p > y_i + r_i$ or $p < y_i - r_i$. But, the latter case cannot hold (otherwise, the exclusive coverage of s_i would be to the right of that of s_j). Since p is covered by s_j, we have $p \leq y_j + r_j$. Therefore, $y_i + r_i < p \leq y_j + r_j$. By using a symmetric argument, we can also prove $y_i - r_i < y_j - r_j$ (we omit the details). Clearly, the two inequalities $y_i + r_i < y_j + r_j$ and $y_i - r_i < y_j - r_j$ imply $y_i < y_j$. The observation thus holds.

An interval I of B is called a *left-aligned interval* if the left endpoint of I is at 0 (i.e., I is of the form $[0, x']$ or $[0, x')$). A set of sensors is said to be in *attached positions* if the union of their covering intervals is a continuous interval of the x-axis whose length is equal to the sum of the lengths of these covering intervals. In the sequel, we describe our algorithm.

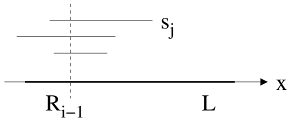


Fig. 1. The set S_{i1} consists of the three sensors whose covering intervals are shown, and $s_{g(i)}$ is s_j

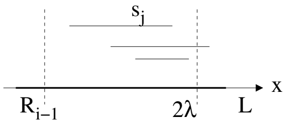


Fig. 2. The set S_{i2} consists of the three sensors whose covering intervals are shown, and $s_{g(i)}$ is s_j if $S_{i1} = \emptyset$

3.1 The Algorithm Description

Initially, we move all sensors of S to the right by the distance λ, i.e., for each $1 \leq i \leq n$, we move s_i to the position $x'_i = x_i + \lambda$. Let C_0 denote the resulting configuration. Clearly, there is a feasible solution for λ if and only if we can move the sensors in C_0 to the left by at most 2λ to form a coverage of B. Thus, henceforth we only need to consider moving the sensors to the left. Recall that we have assumed that the extensions of any two distinct sensors are different; hence in C_0, the extensions of all sensors are also different.

Our algorithm takes a greedy approach. It seeks to find sensors to cover B from left to right, in at most n steps. If $\lambda \geq \lambda^*$, the algorithm will end up with a critical set S^c of sensors along with the destinations for all these sensors.

In step i (initially, $i = 1$), using the configuration C_{i-1} and based on certain criteria, we find a sensor $s_{g(i)}$ and determine its destination $y_{g(i)}$, where $g(i)$ is the index of the sensor in S and $y_{g(i)} \in [x'_{g(i)} - 2\lambda, x'_{g(i)}]$. We then move the sensor $s_{g(i)}$ to $y_{g(i)}$ to obtain a new configuration C_i from C_{i-1} (if $y_{g(i)} = x'_{g(i)}$, C_i is simply C_{i-1}). Let $R_i = y_{g(i)} + r_{g(i)}$ (i.e., the right extension of $s_{g(i)}$ in C_i). Assume $R_0 = 0$. Let $S_i = S_{i-1} \cup \{s_{g(i)}\}$ ($S_0 = \emptyset$ initially). We will show that the sensors in S_i together cover the left-aligned interval $[0, R_i]$. If $R_i \geq L$, we have found a feasible solution with a critical set $S^c = S_i$, and terminate the algorithm. Otherwise, we proceed to step $i + 1$. Further, it is possible that a desired sensor $s_{g(i)}$ cannot be found, in which case we terminate the algorithm and report $\lambda < \lambda^*$. Below we give the details, and in particular, discuss how to determine the sensor $s_{g(i)}$ in each step.

We first discuss a technical issue. Suppose there is a sensor s_t with its right extension at 0 in C_0. We claim s_t cannot be in a critical set of a feasible solution if $\lambda^* \leq \lambda$. Indeed, assume to the contrary that s_t is in a critical set S^c. Then $p(0)$ is the only point on B that can be covered by s_t. Since $L > 0$, there must be another sensor in S^c that also covers $p(0)$ (otherwise, no sensor in S^c would cover the point $p^+(0)$). Hence, s_t is not critical with respect to S^c, a contradiction.

Initially, we have $R_0 = 0$ and $S_0 = \emptyset$. Consider the i-th step of the algorithm with $i \geq 1$. We determine the sensor $s_{g(i)}$, as follows. Define $S_{i1} = \{s_j \mid x'_j - r_j \leq R_{i-1} < x'_j + r_j\}$ (see Fig. 1), i.e., S_{i1} is the set of sensors covering the point $p^+(R_{i-1})$ in the configuration C_{i-1}. Note that any sensor in S_{i1} covers the point $p(R_{i-1})$ in C_{i-1}. If $S_{i1} \neq \emptyset$, we choose the sensor in S_{i1} with the largest right extension as $s_{g(i)}$ and let $y_{g(i)} = x'_{g(i)}$. Otherwise, let S_{i2} be the set of sensors whose left extensions are larger than R_{i-1} and at most $R_{i-1} + 2\lambda$. If $S_{i2} = \emptyset$, we terminate the algorithm and report $\lambda < \lambda^*$. Otherwise, we choose the sensor in S_{i2} with the smallest right extension as $s_{g(i)}$ (e.g., s_j in Fig. 2), and let $y_{g(i)} = R_{i-1} + r_{g(i)}$. If the algorithm is not terminated, we move $s_{g(i)}$ to $y_{g(i)}$ and obtain a new configuration C_i. Let $S_i = S_{i-1} \cup \{s_{g(i)}\}$. Let R_i be the right extension of $s_{g(i)}$ in C_i. If $R_i \geq L$, we have found a feasible solution C_i with the critical set S_i. Otherwise, we proceed to step $i + 1$.

Since there are n sensors in S, the algorithm is terminated in at most n steps.

3.2 The Algorithm Correctness and Implementation

Based on our algorithm description, we have the following lemma.

Lemma 1. *At the end of step i, suppose the algorithm produces the set S_i and the configuration C_i; then S_i and C_i have the following properties. (a) The interval on B covered by the sensors in S_i is $[0, R_i]$. (b) For each $1 < j \leq i$, the right extension of $s_{g(j)}$ is larger than that of $s_{g(j-1)}$. (c) For each $1 \leq j \leq i$, $s_{g(j)}$ is the only sensor in S_i that covers the point $p^+(R_{j-1})$ (with $R_0 = 0$). (d) For each sensor $s_{g(j)} \in S_i$ with $1 \leq j \leq i$, it is either from S_{j1} or S_{j2}. If $s_{g(j)}$ is from S_{j1}, then its position in C_i is the same as that in C_0; otherwise, its left extension is at R_{j-1}, and $s_{g(j)}$ and $s_{g(j-1)}$ are in attached positions if $j > 1$.*

The proof of Lemma 1 is omitted. At its termination, our algorithm either reports $\lambda \geq \lambda^*$ or $\lambda < \lambda^*$. Suppose in step i, our algorithm reports $\lambda \geq \lambda^*$. Then according to the algorithm, it must be $R_i \geq L$. By Lemma 1(a) and (c), C_i is a feasible solution and S_i is a critical set. Further, by Lemma 1(b) and Observation 2, the cover order of the sensors in S_i is $s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}$.

Next, we show that if the algorithm reports $\lambda < \lambda^*$, then there is no feasible solution for λ. This is almost an immediate consequence of Lemma 2.

Lemma 2. *Suppose S'_i is the set of sensors in the configuration C_i whose right extensions are at most R_i. Then the interval $[0, R_i]$ is the largest possible left-aligned interval that can be covered by the sensors of S'_i such that the displacement of each sensor in S'_i is at most λ.*

To prove Lemma 2, the key is to prove the following. If C is a configuration for the sensors of S'_i such that a left-aligned interval $[0, x']$ is covered by the sensors of S'_i, then there always exists a configuration C^* for S'_i in which the interval $[0, x']$ is still covered by the sensors of S'_i and for each $1 \leq j \leq i$, the position of the sensor $s_{g(j)}$ in C^* is $y_{g(j)}$, where $g(j)$ and $y_{g(j)}$ are the values computed by our algorithm. We omit the proof for this.

Suppose our algorithm reports $\lambda < \lambda^*$ in step i. Then according to the algorithm, $R_{i-1} < L$ and both S_{i1} and S_{i2} are $\emptyset$. Let S'_{i-1} be the set of sensors whose right extensions are at most R_{i-1} in C_{i-1}. Since both S_{i1} and S_{i2} are $\emptyset$, no sensor in $S \setminus S'_{i-1}$ can cover any point to the left of the point $p^+(R_{i-1})$ (and including $p^+(R_{i-1})$). By Lemma 2, $[0, R_{i-1}]$ is the largest left-aligned interval that can be covered by the sensors of S'_{i-1}. Hence, the sensors in S cannot cover the interval $[0, p^+(R_{i-1})]$. Due to $R_{i-1} < L$, we have $[0, p^+(R_{i-1})] \subseteq [0, L]$; thus the sensors of S cannot cover $B = [0, L]$. In other words, there is no feasible solution for the distance λ. This establishes the correctness of our algorithm.

We briefly discuss the implementation of our algorithm. Our algorithm needs to maintain two sets of sensors, S_{i1} and S_{i2}. For this purpose, in the preprocessing, we sort the $2n$ extensions of all sensors by the x-coordinate, and move each sensor $s_i \in S$ to x'_i to produce the initial configuration C_0. During the algorithm, we sweep along the x-axis and maintain S_{i1} and S_{i2}, respectively. During the sweeping, we need to perform the sensor insertions and deletions on the two sets. In addition, we need a *search* operation on S_{i1} for finding the sensor in S_{i1} with the largest right extension, and a *search* operation on S_{i2} for finding the sensor in S_{i2} with the smallest right extension. There are $O(n)$ insertion, deletion, and search operations in the entire algorithm.

If we use a balanced binary search tree to store each of these two sets in which the right extensions of the sensors are used as keys, then the algorithm takes $O(n \log n)$ time. Another way is to use an integer data structure (e.g., van Emde Boas tree [9]), as follows. In the preprocessing, we also sort the sensors by their right extensions, and for each sensor, assign the integer k to it as its key if the sensor is the k-th one in the above sorted order. Thus, all such keys form an integer set $\{1, 2, \ldots, n\}$. By using the van Emde Boas tree [9], each operation takes only $O(\log \log n)$ time. Thus, after $O(n \log n)$ time preprocessing, the algorithm takes $O(n \log \log n)$ time for each value λ. Although using the integer data structure does not change the overall running time of our decision algorithm, it helps our optimization algorithm in Section 4 to run faster.

Theorem 1. *After $O(n \log n)$ time preprocessing, for any λ, we can determine whether $\lambda^* \leq \lambda$ in $O(n \log \log n)$ time; further, if $\lambda^* \leq \lambda$, we can compute a feasible solution in $O(n \log \log n)$ time.*

Our optimization algorithm in Section 4 also needs to determine whether λ^* is strictly less than λ (i.e., $\lambda^* < \lambda$) for any λ. By modifying our algorithm for Theorem 1, we have the following Theorem 2 whose proof is omitted.

Theorem 2. *After $O(n \log n)$ time preprocessing, for any value λ, we can determine whether $\lambda^* < \lambda$ in $O(n \log \log n)$ time.*

Theorems 1 and 2 together lead to the following corollary.

Corollary 1. *After $O(n \log n)$ time preprocessing, for any value λ, we can determine whether $\lambda^* = \lambda$ in $O(n \log \log n)$ time.*

4 The Optimization Version of the General BCLS

In this section, we discuss the optimization version of the general BCLS problem. We first give an algorithm overview. In the following discussion, the "decision algorithm" refers to our algorithm for Theorem 1, unless otherwise stated.

Recall that given any value λ, step i of our decision algorithm determines the sensor $s_{g(i)}$ and obtains the set $S_i = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}\}$, in this order, which we also call the *cover order* of the sensors in S_i. In our optimization algorithm, we often use λ as a variable. Thus, $S_i(\lambda)$ (resp., $R_i(\lambda)$, $s_{g(i)}(\lambda)$, and $C_i(\lambda)$) refers to the corresponding S_i (resp., R_i, $s_{g(i)}$, and C_i) obtained by running our decision algorithm on the specific value λ. Denote by C_I the configuration of the input.

Our optimization algorithm takes at most n steps. Step i receives an interval $(\lambda^1_{i-1}, \lambda^2_{i-1})$ and a sensor set $S_{i-1}(\lambda^*)$, with the *algorithm invariants* that $\lambda^* \in (\lambda^1_{i-1}, \lambda^2_{i-1})$ (although we do not know the value λ^*) and for any value $\lambda \in (\lambda^1_{i-1}, \lambda^2_{i-1})$, we have $S_{i-1}(\lambda) = S_{i-1}(\lambda^*)$ and their cover orders are the same. Step i either finds the value λ^* or determines a sensor $s_{g(i)}(\lambda^*)$. The interval $(\lambda^1_{i-1}, \lambda^2_{i-1})$ will shrink to a new interval $(\lambda^1_i, \lambda^2_i) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$ and we also obtain the set $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. Each step can be performed in $O(n \log n \log \log n)$ time. The details of the algorithm are given below.

Initially, let $S_0(\lambda^*) = \emptyset$, $R_0(\lambda^*) = 0$, $\lambda^1_0 = 0$, and $\lambda^2_0 = +\infty$.

Consider a general step i for $i \geq 1$ and we have the interval $(\lambda^1_{i-1}, \lambda^2_{i-1})$ and the set $S_{i-1}(\lambda^*)$. While discussing the algorithm, we will also prove inductively the following lemma about the function $R_i(\lambda)$ with variable $\lambda \in (\lambda^1_i, \lambda^2_i)$.

Lemma 3. *(a) The function $R_i(\lambda)$ for $\lambda \in (\lambda^1_i, \lambda^2_i)$ is a line segment of slope* 1 *or* 0*. (b) We can compute the function $R_i(\lambda)$ for $\lambda \in (\lambda^1_i, \lambda^2_i)$ explicitly in $O(n)$ time. (c) $R_i(\lambda) < L$ for any $\lambda \in (\lambda^1_i, \lambda^2_i)$.*

In the base case for $i = 0$, the statement of Lemma 3 obviously holds. We assume the lemma statement holds for $i - 1$. We will show that after step i with $i \geq 1$, the lemma statement holds for i, and thus the lemma will be proved.

Again, in step i, we need to determine the sensor $s_{g(i)}(\lambda^*)$ and let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. We will also obtain an interval $(\lambda^1_i, \lambda^2_i)$ such that $\lambda^* \in (\lambda^1_i, \lambda^2_i) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$ and for any $\lambda \in (\lambda^1_i, \lambda^2_i)$, $S_i(\lambda) = S_i(\lambda^*)$ holds (with the same cover order). The details are given below. We assume that we already compute explicitly the function $R_{i-1}(\lambda)$ for $\lambda \in (\lambda^1_{i-1}, \lambda^2_{i-1})$, which takes $O(n)$ time by our assumption that the statement of Lemma 3 holds for $i - 1$.

To find the sensor $s_{g(i)}(\lambda^*)$, we first determine the set $S_{i1}(\lambda^*)$. Recall that $S_{i1}(\lambda^*)$ consists of all sensors covering the point $p^+(R_{i-1}(\lambda^*))$ in the configuration $C_{i-1}(\lambda^*)$. For each sensor in $S \setminus S_{i-1}(\lambda^*)$, its position in the configuration $C_{i-1}(\lambda)$ with respect to $\lambda \in (\lambda^1_{i-1}, \lambda^2_{i-1})$ is a function of slope 1. As λ increases in $(\lambda^1_{i-1}, \lambda^2_{i-1})$, by our assumption that Lemma 3(a) holds for $i-1$, the function $R_{i-1}(\lambda)$ is a line segment of slope 1 or 0. If $R_{i-1}(\lambda)$ is of slope 1, then the relative position of $R_{i-1}(\lambda)$ in $C_{i-1}(\lambda)$ does not change and thus the set $S_{i1}(\lambda)$ does not change; if the function $R_{i-1}(\lambda)$ is of slope 0, then the relative position of $R_{i-1}(\lambda)$ in $C_{i-1}(\lambda)$ is monotonically moving to the left. Hence, there are $O(n)$ values for λ in $(\lambda^1_{i-1}, \lambda^2_{i-1})$ that can incur some changes to the set $S_{i1}(\lambda)$ and each such value corresponds to a sensor extension; further, these values can be easily determined in $O(n \log n)$ time by a simple sweeping process (we omit the discussion of it). Let Λ_{i1} be the set of all these λ values. Let Λ_{i1} also contain both λ^1_{i-1} and λ^2_{i-1}, and thus, λ^1_{i-1} and λ^2_{i-1} are the smallest and largest values in Λ_{i1}, respectively. We sort the values in Λ_{i1}. For any two consecutive values $\lambda_1 < \lambda_2$ in the sorted Λ_{i1}, the set $S_{i1}(\lambda)$ for any $\lambda \in (\lambda_1, \lambda_2)$ is the same. By using binary search on the sorted Λ_{i1} and our decision algorithm, we determine (in $O(n \log n \log \log n)$

time) the two consecutive values λ_1 and λ_2 in Λ_{i1} such that $\lambda_1 < \lambda^* \leq \lambda_2$. Further, by Corollary 1, we determine whether $\lambda^* = \lambda_2$. If $\lambda^* = \lambda_2$, then we are done. Otherwise, based on our discussion above, $S_{i1}(\lambda^*) = S_{i1}(\lambda)$ for any $\lambda \in (\lambda_1, \lambda_2)$. Thus, to compute $S_{i1}(\lambda^*)$, we can pick an arbitrary λ in (λ_1, λ_2) and find $S_{i1}(\lambda)$ in the same way as in our decision algorithm. Hence, $S_{i1}(\lambda^*)$ can be easily found in $O(n \log n)$ time. Note that $\lambda^* \in (\lambda_1, \lambda_2) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$.

If $S_{i1}(\lambda^*) \neq \emptyset$, then $s_{g(i)}(\lambda^*)$ is the sensor in $S_{i1}(\lambda^*)$ with the largest right extension. An obvious observation is that for any $\lambda \in (\lambda_1, \lambda_2)$, the sensor in $S_{i1}(\lambda^*)$ with the largest right extension is the same, which can be easily found. We let $\lambda^1_i = \lambda_1$ and $\lambda^2_i = \lambda_2$. Let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. The algorithm invariants hold. Further, as λ increases in (λ_1, λ_2), the right extension of $s_{g(i)}(\lambda)$, which is $R_i(\lambda)$, increases by the same amount. That is, the function $R_i(\lambda)$ on (λ_1, λ_2) is a line segment of slope 1. Therefore, we can compute $R_i(\lambda)$ on (λ_1, λ_2) explicitly in constant time. This also shows Lemma 3(a) and (b) hold for i.

Because the function $R_i(\lambda)$ on (λ_1, λ_2) is a line segment of slope 1, there are three cases depending on the values $R_i(\lambda)$ and L: (1) $R_i(\lambda) < L$ for any $\lambda \in (\lambda_1, \lambda_2)$, (2) $R_i(\lambda) > L$ for any $\lambda \in (\lambda_1, \lambda_2)$, and (3) there exists a unique value $\lambda' \in (\lambda_1, \lambda_2)$ such that $R_i(\lambda') = L$. For Case (1), we proceed to the next step, along with the interval $(\lambda^1_i, \lambda^2_i)$. Clearly, the algorithm invariants hold and Lemma 3(c) holds for i. For Case (2), the next lemma shows that it actually cannot happen due to $\lambda^* \in (\lambda_1, \lambda_2)$.

Lemma 4. *It is not possible that $R_i(\lambda) > L$ for any $\lambda \in (\lambda_1, \lambda_2)$.*

Proof. Assume to the contrary that $R_i(\lambda) > L$ for any $\lambda \in (\lambda_1, \lambda_2)$. Since $\lambda^* \in (\lambda_1, \lambda_2)$, let λ'' be any value in (λ_1, λ^*). Due to $\lambda'' \in (\lambda_1, \lambda_2)$, we have $R_i(\lambda'') > L$. But this would implies that we have found a feasible solution where the displacement of each sensor is at most $\lambda'' \leq \lambda^*$, incurring contradiction.

For the Case (3), since $R_i(\lambda)$ on (λ_1, λ_2) is a line segment of slope 1, we can determine in constant time the unique value $\lambda' \in (\lambda_1, \lambda_2)$ such that $R_i(\lambda') = L$. Clearly, $\lambda^* \leq \lambda'$. By Corollary 1, we determine whether $\lambda^* = \lambda'$. If $\lambda^* = \lambda'$, then we are done; otherwise, we have $\lambda^* \in (\lambda_1, \lambda')$ and update λ^2_i to λ'. We proceed to the next step, along with the interval $(\lambda^1_i, \lambda^2_i)$. Again, the algorithm invariants hold and Lemma 3(c) holds for i.

If $S_{i1}(\lambda^*) = \emptyset$, then we need to compute $S_{i2}(\lambda^*)$. For any $\lambda \in (\lambda_1, \lambda_2)$, the set $S_{i2}(\lambda)$ consists of all sensors whose left extensions are larger than $R_{i-1}(\lambda)$ and at most $R_{i-1}(\lambda) + 2\lambda$ in the configuration $C_{i-1}(\lambda)$. Recall that the function $R_{i-1}(\lambda)$ on $(\lambda^1_{i-1}, \lambda^2_{i-1})$ is linear with slope 1 or 0. Due to $(\lambda_1, \lambda_2) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$, the linear function $R_{i-1}(\lambda) + 2\lambda$ on (λ_1, λ_2) is of slope 3 or 2. Again, as λ increases, the position of each sensor in $S \setminus S_{i-1}(\lambda^*)$ in $C_{i-1}(\lambda)$ is a linear function of slope 1. Therefore, there are $O(n)$ λ values in (λ_1, λ_2) each of which incurs some change to the set $S_{i2}(\lambda)$ and each such λ value corresponds to a sensor extension. Further, these values can be easily determined in $O(n \log n)$ time by a sweeping process (we omit the discussion for this). (Actually, as λ increases, the size of the set $S_{i2}(\lambda)$ is monotonically increasing.) Let Λ_{i2} denote the set of these λ values, and let Λ_{i2} contain both λ_1 and λ_2. Again, $|\Lambda_{i2}| = O(n)$. We sort the values in Λ_{i2}. Using binary search on the sorted Λ_{i2} and our decision algorithm, we determine (in $O(n \log n \log \log n)$ time) the two consecutive values λ'_1 and λ'_2 in Λ_{i2} such that $\lambda'_1 < \lambda^* \leq \lambda'_2$. Further, by Corollary 1, we determine whether $\lambda^* = \lambda'_2$. If $\lambda^* = \lambda'_2$, then we are done. Otherwise, $S_{i2}(\lambda^*) = S_{i2}(\lambda)$ for any $\lambda \in (\lambda'_1, \lambda'_2)$, which can be easily found. Note that $\lambda^* \in (\lambda'_1, \lambda'_2) \subseteq (\lambda_1, \lambda_2)$.

After obtaining $S_{i2}(\lambda^*)$, $s_{g(i)}(\lambda^*)$ is the sensor in $S_{i2}(\lambda^*)$ with the smallest right extension. As before, the sensor in $S_{i2}(\lambda)$ with the smallest right extension is the same for any $\lambda \in (\lambda'_1, \lambda'_2)$. Thus, $s_{g(i)}(\lambda^*)$ can be easily determined. We let $\lambda_i^1 = \lambda'_1$ and $\lambda_i^2 = \lambda'_2$. Let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. The algorithm invariants hold. Further, we examine the function $R_i(\lambda)$, i.e., the right extension of $s_{g(i)}(\lambda)$ in the configuration $C_i(\lambda)$, as λ increases in (λ'_1, λ'_2). Since $s_{g(i-1)}(\lambda^*)$ and $s_{g(i)}(\lambda^*)$ are always in attached positions in this case, for any $\lambda \in (\lambda'_1, \lambda'_2)$, we have $R_i(\lambda) = R_{i-1}(\lambda) + 2r_{g(i)}$. Thus, the function $R_i(\lambda)$ is a vertical shift of $R_{i-1}(\lambda)$ by the distance $2r_{g(i)}$. Because we already know explicitly the function $R_{i-1}(\lambda)$ for $\lambda \in (\lambda'_1, \lambda'_2)$, which is a line segment of slope 1 or 0, the function $R_i(\lambda)$ can be computed in constant time, which is also a line segment of slope 1 or 0. Note that this shows that Lemma 3(a) and (b) hold for i.

Similarly to the case when $S_{i1}(\lambda^*) \neq \emptyset$, since the function $R_i(\lambda)$ in (λ'_1, λ'_2) is a line segment of slope 1 or 0, there are three cases depending on the values $R_i(\lambda)$ and L: (1) $R_i(\lambda) < L$ for any $\lambda \in (\lambda'_1, \lambda'_2)$, (2) $R_i(\lambda) > L$ for any $\lambda \in (\lambda'_1, \lambda'_2)$, and (3) there exists a unique value $\lambda'' \in (\lambda'_1, \lambda'_2)$ such that $R_i(\lambda'') = L$. For Case (1), we proceed to the next step, along with the interval $(\lambda_i^1, \lambda_i^2)$. Clearly, the algorithm invariants hold and Lemma 3(c) holds for i. Similarly to Lemma 4, Case (2) cannot happen due to $\lambda^* \in (\lambda'_1, \lambda'_2)$. For the Case (3), since $R_i(\lambda)$ in (λ'_1, λ'_2) is a line segment of slope 1 or 0, we can compute in constant time the unique value $\lambda'' \in (\lambda'_1, \lambda'_2)$ such that $R_i(\lambda'') = L$. Clearly, $\lambda^* \leq \lambda''$. By Corollary 1, we determine whether $\lambda^* = \lambda''$. If $\lambda^* = \lambda''$, we are done; otherwise, we have $\lambda^* \in (\lambda'_1, \lambda'')$ and update λ_i^2 to λ''. We proceed to the next step, along with the interval $(\lambda_i^1, \lambda_i^2)$. Again, the algorithm invariants and Lemma 3(c) hold for i.

This finishes the discussion of step i of our algorithm. Note that in each case where we proceed to the next step, Lemma 3 holds for i, and thus Lemma 3 has been proved. The running time of step i is clearly bounded by $O(n \log n \log \log n)$.

In at most n steps, the algorithm will stop and find the value λ^*. Then by applying our decision algorithm on $\lambda = \lambda^*$, we finally produce an optimal solution in which the displacement of every sensor is at most λ^*. Since each step takes $O(n \log n \log \log n)$ time, the total time of the algorithm is $O(n^2 \log n \log \log n)$.

Theorem 3. *The general BCLS problem is solvable in* $O(n^2 \log n \log \log n)$ *time.*

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: A survey. Computer Networks 38(4), 393–422 (2002)
2. Bhattacharya, B., Burmester, B., Hu, Y., Kranakis, E., Shi, Q., Wiese, A.: Optimal movement of mobile sensors for barrier coverage of a planar region. Theoretical Computer Science 410(52), 5515–5528 (2009)

3. Chen, A., Kumar, S., Lai, T.: Designing localized algorithms for barrier coverage. In: Proc. of the 13th Annual ACM International Conference on Mobile Computing and Networking, pp. 63–73 (2007)
4. Chen, D., Tan, X., Wang, H., Wu, G.: Optimal point movement for covering circular regions. arXiv:1107.1012v1 (2011)
5. Chen, D., Wang, C., Wang, H.: Representing a functional curve by curves with fewer peaks. Discrete and Computational Geometry (DCG) 46(2), 334–360 (2011)
6. Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. Journal of the ACM 34(1), 200–208 (1987)
7. Czyzowicz, J., Kranakis, E., Krizanc, D., Lambadaris, I., Narayanan, L., Opatrny, J., Stacho, L., Urrutia, J., Yazdani, M.: On Minimizing the Maximum Sensor Movement for Barrier Coverage of a Line Segment. In: Ruiz, P.M., Garcia-Luna-Aceves, J.J. (eds.) ADHOC-NOW 2009. LNCS, vol. 5793, pp. 194–212. Springer, Heidelberg (2009)
8. Czyzowicz, J., Kranakis, E., Krizanc, D., Lambadaris, I., Narayanan, L., Opatrny, J., Stacho, L., Urrutia, J., Yazdani, M.: On Minimizing the Sum of Sensor Movements for Barrier Coverage of a Line Segment. In: Nikolaidis, I., Wu, K. (eds.) ADHOC-NOW 2010. LNCS, vol. 6288, pp. 29–42. Springer, Heidelberg (2010)
9. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Theory of Computing Systems 10(1), 99–127 (1977)
10. Hu, S.: 'Virtual Fence' along border to be delayed. Washington Post (February 28, 2008)
11. Kumar, S., Lai, T., Arora, A.: Barrier coverage with wireless sensors. Wireless Networks 13(6), 817–834 (2007)
12. Li, M., Sun, X., Zhao, Y.: Minimum-Cost Linear Coverage by Sensors with Adjustable Ranges. In: Cheng, Y., Eun, D.Y., Qin, Z., Song, M., Xing, K. (eds.) WASA 2011. LNCS, vol. 6843, pp. 25–35. Springer, Heidelberg (2011)
13. Li, X., Frey, H., Santoro, N., Stojmenovic, I.: Localized sensor self-deployment with coverage guarantee. ACM SIGMOBILE Mobile Computing and Communications Review 12(2), 50–52 (2008)
14. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. Journal of the ACM 30(4), 852–865 (1983)
15. Mehrandish, M., Narayanan, L., Opatrny, J.: Minimizing the number of sensors moved on line barriers. In: Proc. of IEEE Wireless Communications and Networking Conference (WCNC), pp. 653–658 (2011)
16. Tan, X., Wu, G.: New Algorithms for Barrier Coverage with Mobile Sensors. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) FAW 2010. LNCS, vol. 6213, pp. 327–338. Springer, Heidelberg (2010)
17. Yang, S., Li, M., Wu, J.: Scan-based movement-assisted sensor deployment methods in wireless sensor networks. IEEE Trans. Parallel Distrib. Syst. 18(8), 1108–1121 (2007)
18. Zou, Y., Chakrabarty, K.: A distributed coverage and connectivity-centric technique for selecting active nodes in wireless sensor networks. IEEE Trans. Comput. 54(8), 978–991 (2005)

Annotating Simplices with a Homology Basis and Its Applications*

Oleksiy Busaryev[1], Sergio Cabello[2], Chao Chen[3],
Tamal K. Dey[1], and Yusu Wang[1]

[1] Department of Computer Science and Engineering,
The Ohio State University, Columbus, OH 43210, USA
{busaryev,tamaldey,yusu}@cse.ohio-state.edu
[2] Department of Mathematics, University of Ljubljana, Slovenia
sergio.cabello@fmf.uni-lj.si
[3] Institute of Science and Technology Austria, Klosterneuburg, Austria
chao.chen@ist.ac.at

Abstract. Let $\mathcal{K}$ be a simplicial complex and g the rank of its p-th homology group $\mathsf{H}_p(\mathcal{K})$ defined with $\mathbb{Z}_2$ coefficients. We show that we can compute a basis H of $\mathsf{H}_p(\mathcal{K})$ and annotate each p-simplex of $\mathcal{K}$ with a binary vector of length g with the following property: the annotations, summed over all p-simplices in any p-cycle z, provide the coordinate vector of the homology class $[z]$ in the basis H. The basis and the annotations for all simplices can be computed in $O(n^\omega)$ time, where n is the size of $\mathcal{K}$ and $\omega < 2.376$ is a quantity so that two $n \times n$ matrices can be multiplied in $O(n^\omega)$ time. The precomputed annotations permit answering queries about the independence or the triviality of p-cycles efficiently.

Using annotations of edges in 2-complexes, we derive better algorithms for computing optimal basis and optimal homologous cycles in 1 - dimensional homology. Specifically, for computing an optimal basis of $\mathsf{H}_1(\mathcal{K})$, we improve the previously known time complexity from $O(n^4)$ to $O(n^\omega + n^2 g^{\omega-1})$. Here n denotes the size of the 2-skeleton of $\mathcal{K}$ and g the rank of $\mathsf{H}_1(\mathcal{K})$. Computing an optimal cycle homologous to a given 1-cycle is NP-hard even for surfaces and an algorithm taking $2^{O(g)} n \log n$ time is known for surfaces. We extend this algorithm to work with arbitrary 2-complexes in $O(n^\omega) + 2^{O(g)} n^2 \log n$ time using annotations.

Keywords: Simplicial complex, topology, homology basis, optimal cycles, matrix multiplication.

1 Introduction

Cycles play a fundamental role in summarizing the topological information about the underlying space that a simplicial complex represents. Homology groups are

* Research was partially supported by the Slovenian Research Agency, program P1-0297 and NSF grant CCF 1064416.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 189–200, 2012.

well known algebraic structures that capture topology of a space by identifying equivalence classes of cycles. Consequently, questions about homological characterizations of input cycles often come up in computations dealing with topology. For example, to compute a shortest basis of a homology group with a greedy approach, one has to test several times whether the cycles in a given set are *independent*. To determine the topological complexity of a given cycle, a first level test could be deciding if it is *null homologous*, or equivalently if it is a boundary. Recently, a number of studies have been done that concern with the computation of such topological properties of cycles [1,2,3,4,5,6].

Two optimization questions about cycles have caught the attention of researchers because of their relevance in applications: (i) compute an optimal homology basis, i.e. a set of cycles that form a basis of the corresponding homology group and whose weight is minimum among all such basis; (ii) compute an optimal homologous cycle, which asks to compute a cycle with minimum weight in the homology class of a given cycle. Chen and Freedman [3] have shown that both problems for p-dimensional cycles, p-cycles in short, with $p > 1$, are NP-hard to approximate within constant factor. Thus, it is not surprising that most of the studies have focused on 1-cycles except for a special case considered in [7]. In this paper, we use *simplex annotations* which lead to better solutions to these problems. We only consider homology over the field $\mathbb{Z}_2$.

Annotation. An *annotation* for a p-simplex is a length g binary vector, where g is the rank of the p-dimensional homology group. These annotations, when summed up for simplices in a given cycle z, provide the coordinate vector of the homology class of z in a pre-determined homology basis. Such coordinates are only of length g and thus help us determine efficiently the topological characterization of z. We provide an algorithm to compute such annotations in $O(n^{\omega})$ time, where n is the number of input simplices and ω is a quantity so that two $n \times n$ matrices can be multiplied in $O(n^{\omega})$ time. It is known that ω is smaller than 2.376 [8].

The high-level idea for computing the annotation can be described as follows. We first compute an appropriate basis Z of the cycle space $\mathsf{Z}_p(\mathcal{K})$ with the following property: each cycle $z \in Z$ has a *sentinel*, which is a simplex σ_z that appears in the cycle z and in no other cycle from Z. We can then express any cycle z_0 efficiently in the basis Z as the addition of the cycles z from Z whose sentinels σ_z are contained in z_0. Next, we compute an arbitrary homology basis H of $\mathsf{H}_p(\mathcal{K})$. The annotation of any non-sentinel simplex is simply 0, while the annotation of a sentinel σ_z is the coordinates of the homology class $[z]$ in the basis H. Because of linearity, the sum of the annotations over the sentinels in a cycle gives the homology class of that cycle. We show how matrix decomposition algorithms can be leveraged to compute these bases and annotations efficiently.

Annotations have been used extensively to work with 1-dimensional homology in surfaces, where they can actually be computed in linear time; see for example [1,5,6]. However, we are not aware of previous works using annotations to work with homology in higher dimensions or in general simplicial complexes.

For readers familiar with cohomology, it may be worth pointing out that cocycles $\{\phi_i\}_{i=1\ldots g}$ whose classes generate the cohomology group provide an annotation by assigning the binary vector $(\phi_1(\sigma), \ldots, \phi_g(\sigma))$ to simplex σ. From this viewpoint, annotations can be seen as exposing the classical relation between homology and cohomology groups.

Applications. Our annotation technique has the following applications.

1. Using the annotations for edges, we can compute an optimal basis for 1-dimensional homology group $\mathsf{H}_1(\mathcal{K})$ in $O(n^\omega + n^2 g^{\omega-1})$ time, where g is the first Betti number of a simplicial complex $\mathcal{K}$. This improves the previous $O(n^4)$ best known algorithm for computing an optimal homology basis in simplicial complexes [4].
2. Since computing an optimal homologous cycle is NP-hard even for 1-cycles [2,9] in surfaces, Chambers et al. [2] designed an algorithm taking near-linear time when g is constant. Erickson and Nayyeri [5] improved the running time to $2^{O(g)} n \log n$, and Italiano et al. [10] provided an $g^{O(g)} n \log\log n$ algorithm. Using our annotations together with the approach of Erickson and Nayyeri, we obtain an algorithm for finding an optimal homologous cycle in simplicial complexes in $O(n^\omega) + 2^{O(g)} n^2 \log n$ time.
3. Using annotations for p-simplices, we can determine if a given p-cycle is null homologous or if two p-cycles are homologous in time $O(tg)$ time where $t > p$ is the number of p-simplices in the given p-cycles. Given a set of p-cycles, we can also answer queries about their homology independence. A set of p-cycles is called *homology independent* if they represent a set of linearly independent homology classes. A maximal subset of homology independent cycles from a given set of k cycles with t simplices can be computed in $O(tg + (g+k)g^{\omega-1})$ time after computing the annotations.

In many applications, g, the dimension of the concerned homology group is small and can be taken as a constant. In such cases, the applications listed above benefit considerably, e.g., applications in 1 and 2 run in $O(n^\omega)$ time.

2 Background

Homology. In this paper, we focus on simplicial homology over the field $\mathbb{Z}_2$; see comments in the conclusion section for extension to other finite fields. We briefly introduce the notations for chains, cycles, boundaries, and homology groups of a simplicial complex, adapted to $\mathbb{Z}_2$.

Let $\mathcal{K}$ be a simplicial complex. Henceforth, we assume that $\mathcal{K}$ is connected and use $\mathcal{K}_p$ to denote the set of simplices in $\mathcal{K}$ of dimension at most p. To work with the 1-skeleton we use $V = \mathcal{K}_0$, $E = \mathcal{K}_1$, and borrow standard notation from graph theory.

A *p-chain* in $\mathcal{K}$ is a formal sum of p-simplices, $c = \sum_{\sigma \in \mathcal{K}_p} \alpha_\sigma \sigma$, $\alpha_\sigma \in \mathbb{Z}_2$. The set of p-chains forms a vector space $\mathsf{C}_p(\mathcal{K})$ under $\mathbb{Z}_2$-addition where the empty chain plays the role of identity 0. The chain group C_p is in one-to-one correspondence to the family of subsets of $\mathcal{K}_p$. Hence C_p is isomorphic to the n_p-dimensional binary vector space $(\mathbb{Z}_2)^{n_p}$, where n_p is the number of p-simplices in $\mathcal{K}$. A natural basis of C_p consists of the p-simplices in $\mathcal{K}$. In this basis, the coordinate vector of a p-chain is the incidence vector telling which p-simplices appear in the corresponding subset.

The boundary of a p-simplex is a $(p-1)$-chain consisting of the set of its $(p-1)$-faces. This can be linearly extended to a *boundary map* $\partial_p \colon \mathsf{C}_p \to \mathsf{C}_{p-1}$, where the boundary of a chain is defined as the sum of the boundaries of its elements. Using the natural bases of C_p and C_{p-1}, computing the boundary of a p-chain corresponds to multiplying the chain vector with a boundary matrix $[b_1\ b_2 \cdots b_{n_p}]$ whose column vectors are boundaries of p-simplices. We slightly abuse the notation and denote the *boundary matrix* also with ∂_p.

We define the *group of p-cycles* as the kernel of ∂_p, $\mathsf{Z}_p := \ker \partial_p$, and define the *group of p-boundaries* as the image of ∂_{p+1}, $\mathsf{B}_p := \operatorname{im} \partial_{p+1}$. The latter is a subgroup of the former. The *p-th homology group* H_p is the quotient $\mathsf{Z}_p/\mathsf{B}_p$. Each element in H_p, called a *homology class*, is an equivalence class of p-cycles whose differences are p-boundaries. For any p-cycle z, we use $[z]$ to denote the corresponding homology class. Two cycles are *homologous* when they belong to the same homology class. Note that Z_p, B_p, and H_p are also vector spaces. We call their bases a *cycle basis*, a *boundary basis*, and a *homology basis* respectively. The dimension of the p-th homology group is called the *p-th Betti number*. We will denote it by g. A set of p-cycles $\{z_1, \cdots, z_g\}$ is a *homology cycle basis* if the set of classes $\{[z_1], \cdots, [z_g]\}$ forms a homology basis.

Optimization problems. Given a simplicial complex $\mathcal{K}$, exponentially many cycles may belong to a homology class $[z]$. We consider an optimization problem over such a set with all p-cycles assigned weights. Given a real weight $w(\sigma) \geq 0$ for each p-simplex σ, we define the weight of a cycle as the sum of the weights of its simplices, $w(z) = \sum_{\sigma \in z} w(\sigma)$. For example, when $p = 1$ and the weights are the lengths of the edges, the weight of a cycle is its length and the optimization problem seeks for the shortest cycle in a given class. Formally, we state:

Problem 1. Given a simplicial complex and a cycle z, find $\operatorname{argmin}_{z_0 \in [z]} w(z_0)$.

Next, we consider an optimization problem over the set of all homology cycle bases. The weight of a homology cycle basis H is defined as the sum of the weights of its elements, $w(H) = \sum_{z \in H} w(z)$. Note that a simplex may contribute to the weight multiple times if it belongs to multiple cycles in the basis H. Formally, we have the following problem.

Problem 2. Given a simplicial complex, find a homology cycle basis H with minimal $w(H)$.

3 Efficient Matrix Operations

Under $\mathbb{Z}_2$ coefficients, the groups C_p, Z_p, B_p, and H_p are all vector spaces. Linear maps among such spaces or change of bases within the same space can be represented by matrices and operations on them. Our algorithm computes simplex annotations via manipulations of such matrices and bases. Several of our computations use the following concept.

Definition 1 (Earliest Basis). *Given a matrix A with rank r, the set of columns $B_{opt} = \{a_{i_1}, \cdots, a_{i_r}\}$ is called the* earliest basis *if the column indices $\{i_1, \cdots, i_r\}$ are the lexicographically smallest index set such that the corresponding columns of A have full rank.*

For convenience, we often use the same symbol to denote both a set of vectors and the matrix they form and denote by B_{opt} also the matrix $[a_{i_1}\ a_{i_2} \cdots a_{i_r}]$. It is convenient to consider the following alternative view of the earliest basis: a column vector of A is in the earliest basis if and only if it does not belong to the subspace generated by column vectors to its left.

We next summarize the operations on matrices that we need. For simplicity, we assume that the matrix multiplication exponent $\omega > 2$; otherwise, some additional logarithmic terms appear in the running times. The following proposition follows from results in [11,12,13] and some additional observations narrated in the full version of the paper.

Proposition 1. *Let A be an $m \times n$ matrix of rank r with entries over $\mathbb{Z}_2$ where $m \leq n$.*

(a) *If A is square and has full rank, one can compute A^{-1} in $O(n^\omega)$ time.*
(b) *There is an $O(n^\omega)$ time algorithm to compute the earliest basis B_{opt} of A.*
(c) *In $O(n^\omega)$ time, one can compute the coordinates of all columns of A in B_{opt}. Formally, one can compute $AP = B_{opt}[I_r \mid R]$, where P is a permutation matrix, I_r is an $r \times r$ identity matrix, and R is an $r \times (n - r)$ matrix.*

4 Annotating Edges

Let $\mathcal{K}$ be a given simplicial complex. First, we define annotations in general terms using g for the dimension of $\mathsf{H}_p(\mathcal{K})$.

Definition 2 (Annotations). *An* annotation *for p-simplices is a function $\mathrm{a}\colon \mathcal{K}_p \to (\mathbb{Z}_2)^g$ with the following property: any two p-cycles z and z' belong to the same homology class "if and only if"*

$$\sum_{\sigma \in z} \mathrm{a}(\sigma) \;=\; \sum_{\sigma \in z'} \mathrm{a}(\sigma).$$

The annotation *of any p-cycle z is defined by $\mathrm{a}(z) = \sum_{\sigma \in z} \mathrm{a}(\sigma)$.*

We will construct annotations using coordinate vectors of cycles in a homology basis. Let $H = (h_1, h_2, \ldots, h_g)$ be a basis of the vector space $\mathsf{H}_p(\mathcal{K})$. For a p-cycle z, if $[z] = \sum_{i=1}^{g} \lambda_i h_i$ where each $\lambda_i \in \mathbb{Z}_2$, then the coordinate vector of $[z]$ in H is $(\lambda_1, \ldots, \lambda_g) \in (\mathbb{Z}_2)^g$. The question is how to annotate the p-simplices so that the sum of annotations in the simplices of z gives $(\lambda_1, \ldots, \lambda_g)$.

In this section, we explain the technique for annotating edges. An extension to p-simplices is explained in Section 6. We compute edge annotations in three steps. First, we construct a cycle basis Z in which any cycle can be expressed in simple and efficient terms. Second, we find a homology cycle basis H. Last, we compute the homology of each cycle in Z in the homology cycle basis H. From this information, one can compute the homology class of any other cycle using vector sums in the coordinate system provided by H. The approach is based on using a spanning tree of the 1-skeleton to generate the space of cycles. The approach of using a spanning tree to generate the fundamental and the homology group is well known in topology.

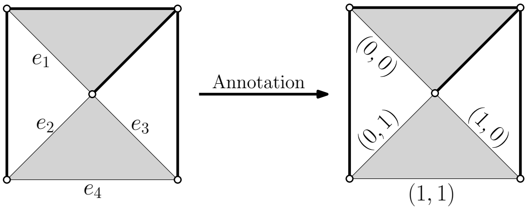
Step 1: Computing a cycle basis $\boldsymbol{Z}$*.* Let us fix throughout this section a spanning tree T in the 1-skeleton of $\mathcal{K}$; it contains $n_0 - 1$ edges. Let $k = n_1 - n_0 + 1$ be the number of edges in $E \setminus E(T)$. We fix an enumeration $e_1, \ldots, e_{n_1}$ of the edges of E with the property that the edges $e_1, \ldots, e_k$ are precisely the edges of $E \setminus E(T)$. Thus, $e_{k+1}, \ldots, e_{n_1}$ are the edges of T. The edges of $E \setminus E(T)$ are called *sentinel edges*, while the edges of $E(T)$ are *non-sentinel edges.*

For any sentinel edge $e \in E \setminus E(T)$, denote by $\gamma(T, e)$ the cycle corresponding to the unique simple path that connects the endpoints of e in T plus the edge e. We call it a *sentinel cycle.* Let Z be the set of such sentinel cycles $\{\gamma(T, e_1), \gamma(T, e_2), \ldots, \gamma(T, e_k)\}$. We have the following property: a sentinel edge e_i belongs to a sentinel cycle $\gamma(T, e_j)$ if and only if $i = j$. For completeness, we set $\gamma(T, e) = 0$ when e belongs to T. The following result is probably folklore.

Proposition 2 (Cycle basis). *Z is a cycle basis and for any cycle $z \in \mathsf{Z}_1$ we have $z = \sum_{e \in z} \gamma(T, e)$.*

Step 2: Computing a homology cycle basis $\boldsymbol{H}$*.* In this step, we compute a homology cycle basis H from Z with the help of Proposition 1(b). Specifically, we construct a new matrix $[\partial_2 \mid Z]$ with the submatrix Z being formed by the chain vectors of cycles in Z. We compute the earliest basis $\widetilde{Z} = [B \mid H]$ of $[\partial_2 \mid Z]$ where B contains the first $r = \mathrm{rank}(\partial_2)$ columns of $\widetilde{Z}$. Since the set of columns of ∂_2 generates the boundary group, by the definition of earliest basis, it is necessary that the columns in B come from ∂_2 and form a boundary cycle basis. Since Z and hence $\partial_2 \cup Z$ generates the cycle group, the remaining columns of $\widetilde{Z}$, namely H, form a homology cycle basis.

Step 3: Computing annotations. Finally, for elements of Z we compute their coordinates in the cycle basis $\widetilde{Z}$. For each sentinel cycle $z = \gamma(T, e)$, we compute its coordinate vector in $\widetilde{Z}$ by solving the linear system $\widetilde{Z}x = z$. The last g entries of x give its coordinates in the basis H. We use this length g vector as the annotation of the sentinel edge e. We can compute annotations for all sentinel edges together by solving $\widetilde{Z}X = Z$ and taking the last g rows of the solution X. For a non-sentinel edge, we set its annotation to be the zero vector.



An example of annotation for a 2-complex is shown on right. The edges of a spanning tree are shown with thicker edges. The edges e_1, e_2, e_3, e_4 are sentinel edges. The cycles given by sentinel edges e_2 and e_3 form the homology cycle basis H computed by the algorithm. We show the annotations for the sentinel edges; all other edges get annotation $(0, 0)$. The annotation of $(1, 1)$ for e_4 makes it possible to evaluate the cycle $e_2e_3e_4$ to $(0, 0)$ as it is null-homologous and also evaluate the outer boundary to $(1, 1)$ as it is homologous to the sum of the two holes.

Theorem 1. *The algorithm above computes an annotation of length* rank $\mathsf{H}_1(\mathcal{K})$ *for the edges of a* 2*-complex* $\mathcal{K}$ *in* $O(n^\omega)$ *time, where* n *is the size of* $\mathcal{K}$*.*

Proof. From Step 3, the annotation of a sentinel edge e is the coordinate vector of the homology class $[\gamma(T, e)]$. It then follows from Proposition 2 that, for any cycle z, the coordinate vector of the homology class $[z]$ is simply the summation of annotations of all edges in z. For the time complexity, notice that Step 1 requires computing a spanning tree and the cycle basis Z, which takes $O(n^2)$ time. Steps 2 and 3 take $O(n^\omega)$ time because of Proposition 1(b) and (a) respectively. □

5 Optimality for 1-Cycles

5.1 Shortest Homology Basis

In this section we discuss the problem of computing an optimal homology basis for one dimensional homology H_1. The optimal homology cycle basis here is the *shortest homology basis* since we minimize the weights / lengths. We present an efficient algorithm that combines the approach of Erickson and Whittlesey [6] and our annotation technique. The approach restricts the search to a well-structured family of cycles, represents each cycle in this family with a length-g binary vector, and then reduces the computation to the problem of finding an earliest basis in a matrix of size $g \times n^2$.

For each vertex $s \in V$, let T_s be the *shortest path tree* from s with respect to the weight function. Denote by Z_s the set of sentinel cycles corresponding to this tree T_s and $\mathit{\Pi}$ the union of Z_s for all $s \in V$, that is,

$$\mathit{\Pi} \;=\; \bigcup_{s \in V} Z_s \;=\; \bigcup_{s \in V} \{\gamma(T_s, e) \mid e \in E \setminus E(T_s)\}.$$

The following property was noted by Erickson and Whittlesey [6]. See also Dey, Sun, and Wang [4] for an extension.

Proposition 3. *If we sort the cycles of* $\mathit{\Pi}$ *in non-decreasing order of their weights, the earliest basis of* $\mathit{\Pi}$ *is a shortest homology basis.*

Theorem 2. *Let $\mathcal{K}$ be a simplicial complex of size n. We can find a shortest homology basis in time $O(n^\omega + n^2 g^{\omega-1})$ where $g = \mathrm{rank}(\mathsf{H}_1(\mathcal{K}))$.*

Proof. By Theorem 1 we compute annotations for all edges in $O(n^\omega)$ time. Let $\mathrm{a}(e)$ be such annotation for an edge e, and $\mathrm{a}(z) = \sum_{e \in z} \mathrm{a}(e)$ for any 1-cycle z.

Next we compute annotations for all cycles $z \in \mathit{\Pi}$. Instead of computing them one by one, we annotate all cycles in Z_s at once for each s. Given a fixed s, we first compute T_s in $O(n \log n)$ time. We assign a g-long label $\ell(x)$ to each vertex $x \in V$ which is the label $\ell(x')$ of its parent x' plus the annotation of the edge xx', $\mathrm{a}(xx')$. We compute labels for all vertices in $O(ng)$ time by a breadth-first traversal of T_s. Afterward, the annotation of any sentinel cycle $\gamma(T_s, xy) \in Z_s$ is computed in $O(g)$ time as $\ell(x) + \ell(y) + \mathrm{a}(xy)$. Thus, we can compute the annotations for all cycles in Z_s in $O(ng)$ time given T_s and edge annotations. To annotate all cycles of $\mathit{\Pi}$, we repeat the procedure for all source vertices s. Computing annotations for all cycles thus takes $O(n^2 g + n^2 \log n)$ time.

Since annotations of cycles give us the homology classes they belong to, we can use them to find a shortest homology basis. We sort cycles in $\mathit{\Pi}$ in non-decreasing order of their weights in $O(n^2 \log n)$ time. Let $z_1, z_2, z_3, \ldots$ be the resulting ordering. We construct a matrix A whose ith column is the vector $\mathrm{a}(z_i)$, and compute its earliest basis. By Proposition 3, the cycles defining the earliest basis of A form a shortest homology basis. Since there are up to n^2 elements in $\mathit{\Pi}$, the matrix A has size $g \times n^2$, and thus it is inefficient to compute its earliest basis using Proposition 1 directly. Instead, we use the following iterative method to compute the set J of indices of columns that define the earliest basis.

We partition A from left to right into submatrices $A = [A_1|A_2|\cdots]$ where each submatrix A_i contains g columns with the possible exception of the last submatrix which contains at most g columns. Initially, we set J to be the empty set. We then iterate over the submatrices A_i by increasing index. At each iteration we compute the earliest basis for the matrix $[A_J|A_i]$ where A_J is the submatrix whose column indices are in J. We then set J to be the indices from the resulting earliest basis, increment i, and proceed to the next iteration. At each iteration we need to compute the earliest basis in a matrix with g rows and at most $|J| + g \leq 2g$ columns. There are at most $O(n^2/g)$ iterations each taking $O(g^\omega)$ time.

We obtain the claimed time bound by adding up the time to annotate edges, annotate cycles in $\mathit{\Pi}$, and compute the earliest basis. □

5.2 Shortest Homologous Cycle

In this section, we show how to compute the shortest cycle in a given one-dimensional homology class. In fact, within the same running time, we can compute a shortest cycle in each homology class. The idea is to use covering graphs, and it closely resembles the approach of Erickson and Nayyeri [5]. We skip most of the details because of this similarity. Nevertheless, our main contribution is the use of the annotations from Section 4.

Let $G = (V, E)$ be the 1-skeleton of $\mathcal{K}$. We first compute an annotation $\mathrm{a}\colon E \to (\mathbb{Z}_2)^g$, as given by Theorem 1. A walk in G is a sequence of vertices $x_0 x_1 \ldots x_t$

connected by edges in E. It is closed if $x_0 = x_t$. In this section we keep using the term cycle for elements of Z_1. Each closed walk in G defines a cycle, where only edges appearing an odd number of times in the walk are kept. The annotation $\mathrm{a}(w)$ of a walk $w = x_0 x_1 \dots x_t$ is defined as the sum of the annotations of its edges $x_{i-1}x_i$ for $i = 1, \dots, t$. Notice that the annotation $\mathrm{a}(w)$ of a closed walk w is the annotation of the cycle defined by w, as annotations in edges appearing an even number of times in the walk cancel out.

We construct a covering graph $\widetilde{G}$ of the 1-skeleton of $\mathcal{K}$, defined as follows:

- $V(\widetilde{G}) = V \times (\mathbb{Z}_2)^g$.
- vertex $(v, h) \in V \times (\mathbb{Z}_2)^g$ is adjacent to $(v', h') \in V \times (\mathbb{Z}_2)^g$ if and only if $e = vv'$ is an edge of E and $h' = h + \mathrm{a}(e)$. The weight of an edge $(v, h)(v', h')$ is the weight of vv'.

The graph $\widetilde{G}$ has $n_0 \cdot 2^g$ vertices and $n_1 \cdot 2^g$ edges. The covering graph for the example shown previously in section 4 for annotation is depicted above. The second coordinate of a vertex $(v, h) \in V(\widetilde{G})$ is used to encode the homology of cycles.

Proposition 4. *For all $h \in (\mathbb{Z}_2)^g$, we can compute a shortest walk w_h in G among all closed walks with annotation h in $O(2^g n^2 (g + \log n))$ time.*

We say that a cycle is *elementary* if it is connected and each vertex is adjacent to at most two edges of the cycle. Each cycle is the union of edge-disjoint elementary cycles. First, we bound the number of elementary cycles in optimal solutions and then use dynamic programming across annotations and the number of elementary cycles to obtain the following (full details appear in the extended version available from the authors' web-pages).

Proposition 5. *The shortest cycle in any given homology class consists of at most g elementary cycles.*

Theorem 3. *In $O(n^\omega + 4^g g + 2^g n^2 (g + \log n)) = O(n^\omega) + 2^{O(g)} n^2 \log n$ time we can compute the shortest homology cycle for all homology classes in H_1.*

6 Annotating p-Simplices

In this section, we show how to compute annotations for p-simplices. Notice that the only thing we need to generalize is the first step: find a set Σ of p-simplices (*sentinel simplices*) with cardinality $\dim(\mathsf{Z}_p)$ and a cycle basis $Z = \{z_\sigma\}_{\sigma \in \Sigma}$ (*sentinel cycles*) for the p-cycle group Z_p with the property that z_σ contains $\sigma' \in \Sigma$ if and only if $\sigma = \sigma'$. With this property, any p-cycle z can then be written as $z = \sum_{\sigma \in z \cap \Sigma} z_\sigma$. Taking $z_\sigma = 0$ for all $\sigma \notin \Sigma$, we have $z = \sum_{\sigma \in z} z_\sigma$. With such a basis, we proceed with Step 2 and 3 just like in the case for edges to annotate p-simplices. Below, we explain how to compute such a cycle basis Z.

In the case for annotating edges, we first fix a spanning tree. The boundaries of its edges form a 0-dimensional boundary basis. Any of the remaining edges when added to the tree creates a unique 1-cycle which is kept associated with this edge as a sentinel cycle. For p-simplices, $p > 1$, we do not have a spanning tree, but Proposition 1(c) provides us an algebraic tool that serves the same purpose.

Specifically, consider the $n_{p-1} \times n_p$ boundary matrix ∂_p of rank r, where the i-th column in ∂_p corresponds to the $(p-1)$-boundary of p-simplex σ_i. Using Proposition 1(c) we can obtain an $n_p \times n_p$ matrix P, an $n_{p-1} \times r$ matrix B_{opt}, and an $r \times (n_p - r)$ matrix R so that

$$\partial_p P = B_{opt}[I_r \mid R].$$

Notice that P permutes the p-simplices so that the first r columns of $\partial_p P$ form the earliest basis B_{opt}. By reordering the columns of ∂_p, we may assume that P is the identity, giving $\partial_p = B_{opt}[I_r \mid R]$. In this scenario, the columns of B_{opt} form a basis of the column-space of ∂_p, and contains the first $r = \text{rank}\,(\partial_p)$ columns of ∂_p. Note that the i-th column in $[I_r \mid R]$ gives the coordinate vector of the boundary cycle for σ_i in the boundary basis B_{opt}.

Take the first r p-simplices $\{\sigma_1, \ldots, \sigma_r\}$. Their boundaries are linearly independent. Therefore, no subset of them can form a p-cycle. In analogy to Section 4, we use T for this collection of p-simplices and call them non-sentinel simplices. The set $\Sigma = \mathcal{K}_p \setminus T$ of p-simplices are the *sentinel simplices.*

Now consider any sentinel p-simplex, say $\sigma_{r+i} \in \Sigma$. Its boundary is the $(r+i)$-th column in ∂_p and is equal to $B_{opt}R[i]$, where $R[i]$ is the i-th column of R. This means that $\partial_p \sigma_{r+i} = \sum_{j=1}^{r} R[j,i](\partial_p \sigma_j)$ where $R[j,i]$ is the j-th entry in the i-th column of R. Hence taking the set of p-simplices σ_j, $j \in [1,r]$, whose corresponding entries $R[j,i]$ are 1, plus σ_{r+i} itself, we obtain a p-cycle $\gamma(T, \sigma_{r+i})$. We call this p-cycle a *sentinel cycle.* Similar to Section 4, we set $\gamma(T,\sigma) = 0$ for each non-sentinel simplex $\sigma \in T$. Clearly, $\gamma(T, \sigma_{r+i})$ can only contain one simplex from Σ which is σ_{r+i}. We have the desired property: a sentinel simplex $\sigma \in \Sigma$ belongs to a sentinel cycle $\gamma(T, \sigma')$ if and only if $\sigma = \sigma'$. Finally, observe that the columns of $\begin{bmatrix} R \\ I_{n_p - r} \end{bmatrix}$ give the set of sentinel cycles Z. The $(n_p - r) \times (n_p - r)$ identity matrix $I_{n_p - r}$ associates each sentinel cycle $\gamma(T,\sigma)$ in Z to its sentinel p-simplex σ. Similar to Proposition 2, we have:

Proposition 6. *$Z = \{\gamma(T, \sigma_{r+1}), \ldots, \gamma(T, \sigma_{n_p})\}$ is a cycle basis, and for any p-cycle z we have $z = \sum_{\sigma \in z} \gamma(T,\sigma)$.*

Combining this proposition with Step 2 and 3 from Section 4, we obtain the following theorem.

Theorem 4. *We can annotate the p-simplices in a simplicial complex with n simplices in $O(n^{\omega})$ time.*

7 Null Homology and Independence

Our annotation algorithm can be used to address some of the computational problems involving p-cycles.

Null homology. A p-cycle z in a simplicial complex $\mathcal{K}$ is called *null homologous* if $[z] = 0$. A cycle is null homologous if and only if it has zero coordinates in *some* and hence *any* basis of $\mathsf{H}_p(\mathcal{K})$. Consider the following two problems:

> Q1: Given a p-cycle z in a simplicial complex $\mathcal{K}$, decide if z is null homologous.
>
> Q2: Given two p-cycles z_1 and z_2 in a simplicial complex $\mathcal{K}$, decide if z_1 and z_2 are homologous.

With annotations whose computations take $O(n^\omega)$ time, we can obtain a query time of $O(tg)$ for Q1 where $g = \dim \mathsf{H}_p(\mathcal{K})$ and t is the number of p-simplices in z. For this we simply add the annotations of the p-simplices in z and check if the result is zero, which takes $O(tg)$ time. The problem Q2 reduces to Q1 because z_1 and z_2 are homologous if and only if $z_1 + z_2$ is null homologous. Therefore, Q2 can be answered in $O((t_1 + t_2)g)$ time after $O(n^\omega)$ time preprocessing where t_1 and t_2 are the number of p-simplices in z_1 and z_2 respectively.

Independence. An analogous problem to testing null homology is the problem of testing independence.

> Q3: Find a maximally independent subset of a given set of p-dimensional homology classes $[z_1], \ldots, [z_k]$ in a simplicial complex $\mathcal{K}$.

A solution to Q3 can be computed using annotations as follows. By Theorem 4, we can compute an annotation $\mathrm{a}(z_i)$ in $O(tg)$ time for all cycles z_is, where t is the number of simplices altogether in all cycles and each $\mathrm{a}(z_i)$ is a length-g vector. Now construct a matrix A whose ith column is the vector $\mathrm{a}(z_i)$. Notice that the earliest basis of A form a maximally independent subset of column vectors in A. Thus the set of cycles z_is corresponding to columns in this earliest basis form a maximally independent subset of the input set of p-cycles. Since A is of size $g \times k$, we have an $O(g^\omega + k^\omega)$ query time, after an $O(n^\omega)$ preprocessing time. We can improve the query time to $O((g + k)g^{\omega-1})$ by using the iterative technique in Theorem 2. Therefore, total time for computing a maximally independent set takes $O(tg + (g + k)g^{\omega-1})$ time after annotations.

8 Conclusions

In this paper, we present an algorithm to annotate p-simplices in a complex so that computations about the homology groups can be done faster. For defining the weights of a cycle we used 1-norm to combine the weights of the individual edges. For defining the weight of a basis we also used 1-norm to combine the weights of the basis cycles. In these problems we can use any other norm to define these weights.

One may wonder why we cannot extend our annotation approach to compute the optimal homology basis or the optimal homologous cycle for higher dimensional cycles. The main bottleneck for finding an optimal basis is that the

Proposition 3 does not generalize to higher dimensional cycles. Given that the problems are NP-hard in high dimensions even for $g = 1$ [3], such extensions cannot exist unless P=NP.

Finally, we point out that one can use any finite field instead of Z_2 for annotations. Since annotations mainly utilize matrix multiplications which remain valid under any field, the annotation algorithm in section 6 remains applicable without any change. However, the optimal cycles in sections 5 and 5.2 require computations of cycles associated with shortest paths which we do not know how to generalize for general fields. Specifically, it is not clear how to obtain results for applying Propositions 3 and 4. This could be a topic of further research.

References

1. Cabello, S., Colin de Verdière, É., Lazarus, F.: Finding cycles with topological properties in embedded graphs. SIAM J. Disc. Math. 25(4), 1600–1614 (2011)
2. Chambers, E., Erickson, J., Nayyeri, A.: Minimum cuts and shortest homologous cycles. In: Proc. ACM Symp. on Computational Geometry (SOCG), pp. 377–385 (2009)
3. Chen, C., Freedman, D.: Hardness results for homology localization. Discrete and Computational Geometry 45(3), 425–448 (2011)
4. Dey, T.K., Sun, J., Wang, Y.: Approximating loops in a shortest homology basis from point data. In: Proc. ACM Symp. on Computational Geometry (SOCG), pp. 166–175 (2010)
5. Erickson, J., Nayyeri, A.: Minimum cuts and shortest non-separating cycles via homology covers. In: Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 1166–1176 (2011)
6. Erickson, J., Whittlesey, K.: Greedy optimal homotopy and homology generators. In: Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 1038–1046 (2005)
7. Dey, T.K., Hirani, A., Krishnamoorthy, B.: Optimal homologous cycles, total unimodularity, and linear programming. SIAM J. Comput. 40, 1026–1044 (2011)
8. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symb. Comput. 9(3), 251–280 (1990)
9. Chen, C., Freedman, D.: Quantifying homology classes. In: Proc. Symp. on Theoretical Aspects of Computer Science (STACS), pp. 169–180 (2008)
10. Italiano, G.F., Nussbaum, Y., Sankowski, P., Wulff-Nilsen, C.: Improved algorithms for min cut and max flow in undirected planar graphs. In: Proc. ACM Symp. on Theory of Computing (STOC), pp. 313–322 (2011)
11. Bunch, J., Hopcroft, J.: Triangular factorization and inversion by fast matrix multiplication. Math. Comp. 28, 231–236 (1974)
12. Ibarra, O., Moran, S., Hui, R.: A generalization of the fast LUP matrix decomposition algorithm and applications. Journal of Algorithms 3(1), 45–56 (1982)
13. Jeannerod, C.: LSP matrix decomposition revisited (2006), http://www.ens-lyon.fr/LIP/Pub/Rapports/RR/RR2006/RR2006-28.pdf

Do Directional Antennas Facilitate in Reducing Interferences?*

Rom Aschner[1], Matthew J. Katz[1], and Gila Morgenstern[2]

[1] Department of Computer Science, Ben-Gurion University
{romas,matya}@cs.bgu.ac.il
[2] Caesarea Rothschild Institute, University of Haifa
gilamor@cri.haifa.ac.il

Abstract. The coverage area of a directional antenna located at point p is a circular sector of angle α, whose orientation and radius can be adjusted. The interference at p, denoted $I(p)$, is the number of antennas that cover p, and the interference of a communication graph $G = (P, E)$ is $I(G) = \max\{I(p) : p \in P\}$. In this paper we address the question in its title. That is, we study several variants of the following problem: What is the minimum interference I, such that for any set P of n points in the plane, representing transceivers equipped with a directional antenna of angle α, one can assign orientations and ranges to the points in P, so that the induced communication graph G is either connected or strongly connected and $I(G) \leq I$.

In the asymmetric model (i.e., G is required to be strongly connected), we prove that $I = O(1)$ for $\alpha < 2\pi/3$, in contrast with $I = \Theta(\log n)$ for $\alpha = 2\pi$, proved by Korman [12]. In the symmetric model (i.e., G is required to be connected), the situation is less clear. We show that $I = \Theta(n)$ for $\alpha < \pi/2$, and prove that $I = O(\sqrt{n})$ for $\pi/2 \leq \alpha \leq 3\pi/2$, by applying the Erdös-Szekeres theorem. The corresponding result for $\alpha = 2\pi$ is $I = \Theta(\sqrt{n})$, proved by Halldórsson and Tokuyama [10].

As in [12] and [10] who deal with the case $\alpha = 2\pi$, in both models, we assign ranges that are bounded by some constant c, assuming that $UDG(P)$ (i.e., the unit disk graph over P) is connected. Moreover, the $O(\sqrt{n})$ bound in the symmetric model reduces to $O(\sqrt{\Delta})$, where Δ is the maximum degree in $UDG(P)$.

1 Introduction

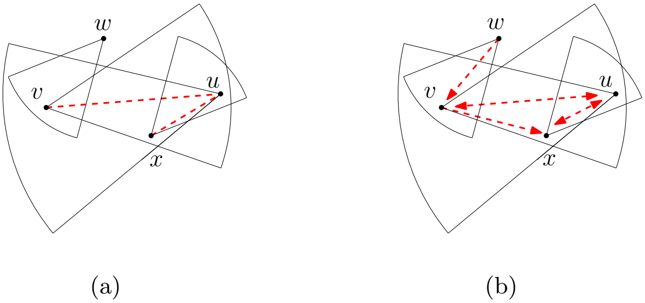
The vast majority of papers dealing with wireless ad-hoc networks, assume that the underlying transceivers are equipped with omni-directional antennas and

* Work by R. Aschner was partially supported by the Lynn and William Frankel Center for Computer Sciences and by the Israel Ministry of Industry, Trade and Labor (consortium CORNET). Work by M.J. Katz was partially supported by the Israel Ministry of Industry, Trade and Labor (consortium CORNET), by grant 1045/10 from the Israel Science Foundation, and by grant 2010074 from the United States – Israel Binational Science Foundation. Work by G. Morgenstern was partially supported by the Lynn and William Frankel Center for Computer Sciences and by the Caesarea Rothschild Institute (CRI).

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 201–212, 2012.

model them by disks. In this paper we consider directional antennas and model them by circular sectors of a fixed angle α. A common belief is that directional antennas facilitate in reducing interferences. This is of course correct for, e.g., a set of independent pairs of transceivers, using the same frequency. In this case, switching to directional antennas can significantly reduce the level of interferences. However, is this still correct for the case where one is required to construct a connected network? In this paper, we study this question.

Let P be a set of n points in the plane, and assume that each point represents a transceiver equipped with a directional antenna. The *coverage area* of a directional antenna located at point p of angle α and range r, is a sector of angle α of the disk of radius r centered at p, where the orientation of the sector can be adjusted. We denote the coverage area of the antenna at p by W_p (since when assuming unbounded range the sector becomes a wedge). The induced *symmetric communication graph* (*sCG*) of P is the undirected graph over P, in which two vertices (i.e., points) u and v are connected by an edge if and only if $v \in W_u$ and $u \in W_v$; see Figure 1(a). A *sCG* over a set P of points is denoted $sCG(P)$. Similarly, the induced *directed communication graph* (*dCG*) (or *asymmetric communication graph*) of P is the directed graph over P, in which there is a directed edge from u to v if and only if $v \in W_u$; see figure 1(b). A *dCG* over a set P of points is denoted $dCG(P)$.



(a) (b)

Fig. 1. Symmetric and asymmetric communication graphs induced by a set of wedges

Most of the papers dealing with directional antennas consider the problem of orienting the antennas and fixing their ranges, such that the induced graph is connected (or strongly connected) and the ranges are not too long. Caragiannis et al. [5] study this problem under the asymmetric model. They show how to orient directional antennas and fix their range, such that the induced graph is strongly connected and the assigned range is minimized. Damian and Flatland [8] and subsequently Bose et al. [3] show how to minimize both the range and the hop-ratio (w.r.t. the unit disk graph), for $\alpha \geq \pi/2$ and for $\alpha > 0$, respectively. Under the symmetric model, Carmi et al. [6] and Ackerman et al. [1] show that it is always possible to obtain a connected graph for $\alpha \geq \pi/3$, assuming the range is unbounded (i.e., equal to the diameter of the underlying point set). Carmi et al. [6] also observe that for $\alpha < \pi/3$ it is not always possible to orient the antennas such that the induced *sCG* is connected. In a companion paper, Aschner et al. [2] show how to minimize both the range and the hop ratio (w.r.t. the unit disk graph) under the symmetric model, for $\alpha \geq \pi/2$.

The efficiency of a wireless network is affected to a large extent by the level of interferences. Interference occurs at a receiver r when two or more transmitters, whose coverage region contains r, transmit simultaneously. Therefore, reducing the level of interferences is a major issue in the study of wireless networks. A few models have been proposed for measuring the level of interferences; see, e.g., [4,11,14]. The most common one is the *receiver-centric* model, where the interference of a node v is the number of nodes that cover it [15].

In many papers dealing with directional antennas it is claimed that directional antennas can reduce the level of interferences (see, e.g., [3,5,13]), but we are not aware of any paper that actually shows the advantage of directional antennas w.r.t interference. In other words, we are not aware of any paper that mathematically establishes this intuitive assertion. The *interference* of a point $v \in P$, denoted $I(v)$, is the number of wedges covering v. I.e., $I(v) = |\{u \in P \setminus \{v\} : v \in W(u)\}|$. In other words it is the in-degree of v in the induced directed communication graph. The *interference* of a communication graph G (whether directed or not) is $I(G) = \max\{I(v) : v \in P\}$.

Von Rickenbach et al. [15] studied the omni-directional symmetric case in 1D (known as the *highway model*). They gave an algorithm that constructs a connected network with interference $O(\sqrt{\Delta})$, where Δ is the maximum degree in $UDG(P)$, and showed that this algorithm approximates the minimum possible interference by a factor of $O(\Delta^{1/4})$. Tan et al. [16] presented an algorithm that solves the problem optimally in sub-exponential time. For points in the plane (i.e., the 2D version) Halldórsson and Tokuyama [10] presented algorithms that construct a network with interference $O(\sqrt{\Delta})$ and ranges at most $\sqrt{2}$ (w.r.t. the unit disk graph). Their construction does not carry to the case of directional antennas since connectivity is not guaranteed. Von Rickenbach et al. [15] showed that even for the 1D version there are instances where interference $\Omega(\sqrt{n})$ is unavoidable. Recently, Korman [12] showed how to obtain a strongly connected network (i.e., the directed case) with $O(\log \Delta)$ interference and ranges at most 1, and showed that $\Omega(\log n)$ interference is sometimes unavoidable.

Our Results. This is the first paper that deals with the problem of constructing (strongly) connected low-interference networks when the underlying transceivers are equipped with directional antennas of some fixed angle α. We study the problem in both the symmetric and asymmetric models. At first glance, it seems natural that the interference should decrease when switching to directional antennas. However, as we show, this is not always the case. We discuss this somewhat unexpected finding in the conclusion section.

In the asymmetric model, we show how to construct a network with interference $O(1)$, for $\alpha < 2\pi/3$ (which is the most interesting range of angles), in contrast with the $\Theta(\log \Delta)$-bound for $\alpha = 2\pi$. Moreover, assuming $UDG(P)$ is connected, we are able to limit the maximum range by an appropriate constant. The problem of constructing a low-interference graph supporting converge-cast operations to a designated node, arises as a subproblem, and in Section 2 we present a solution to it with interference $O(1)$.

In the symmetric model, we obtain several results. First, we show how to construct a network with interference $O(1)$ for points on a line and $\alpha < 2\pi$, in contrast with the $\Theta(\sqrt{\Delta})$-bound for $\alpha = 2\pi$. The 1D construction also applies to the case of a xy-monotone sequence of points, for $\pi/2 \leq \alpha \leq 3\pi/2$. Then, we observe that for $\alpha < \pi/2$ interference $\Omega(n)$ is sometimes unavoidable. Finally, we show how to construct a network with $O(\sqrt{n})$ interference, for $\pi/2 \leq \alpha \leq 3\pi/2$, by applying the Erdös-Szekeres theorem [9]. Moreover, assuming $UDG(P)$ is connected, we are able to reduce the interference to $O(\sqrt{\Delta})$ and limit the maximum range by some constant. The corresponding bound for $\alpha = 2\pi$ is $\Theta(\sqrt{\Delta})$.

2 Constant Interference Converge-Cast

In this section we consider converge-cast with low interference. *Converge-cast* is an operation in which data from many sources is routed towards a single base-station. We show that, using directional antennas, one can construct a converge-cast network with interference $O(1)$. More formally, let P be a set of points in general position in the plane representing transceivers with directional antennas of angle α, and let $s \in P$. We show how to assign to each antenna $p \in P$ an orientation θ and a range r, such that $dCG(P)$ contains a path from each $p \in P$ to s and $I(dCG(P)) = O(1)$.

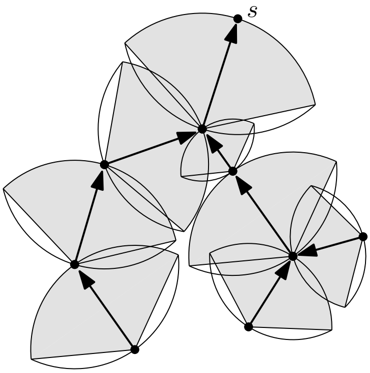


Fig. 2. Constant interference converge-cast

Below, we show how to do this for $\alpha < 2\pi/3$. We actually show that the resulting network has a slightly stronger property. Not only that each point of P lies in a constant number of wedges, but this is also so for any arbitrary point in the plane. More precisely, the depth of the arrangement of the wedges is 5, i.e., every point in the plane is covered by at most 5 wedges.

In order to obtain a dCG with the above properties, we use the Euclidean minimum spanning tree of P, denoted $MST(P)$. First, root $MST(P)$ at s, and orient the antenna at each point $p \in P \setminus \{s\}$ towards its parent $\pi(p)$, i.e., keeping $\pi(p)$ on the bisector of W_p. Next, assign range $|(p, \pi(p))|$ to each antenna $p \in P \setminus \{s\}$. It is easy to see that the resulting dCG contains a path from each $p \in P$ to s; see Figure 2. By Lemma 1 below, any point x in the plane lies in the range of at most 5 antennas. In fact, in the proof of Lemma 1, we consider a different arrangement, whose depth is greater or equal to the depth of the arrangement of wedges.

Notice that each edge (p, q) of $MST(P)$ is associated with a single wedge of range $|(p, q)|$ directed either from p to q or vice versa. For each such edge, denote by $lune\,(p, q)$ the intersection of the two disks centered at p and q of radius $|(p, q)|$. Notice that $lune\,(p, q)$ is just the union of two wedges of angle $2\pi/3$ and range $|(p, q)|$, directed from p to q and from q to p, respectively. Thus, $lune\,(p, q)$ contains the wedge that is associated with the edge (p, q); see Figure 2 for an illustration. Denote the set of all lunes corresponding to the edges of $MST(P)$ by $\mathcal{L}\,(MST(P))$. It is enough to show that the depth of the arrangement of $\mathcal{L}\,(MST(P))$ is at most 5. The proof is based on a similar proof from [7].

We use the following easy claim from [7], which is based on known properties of minimum spanning trees.

Claim. ([7]) Let x be a point in the plane, $x \notin P$. If e is an edge of $MST(P \cup \{x\})$ but not of $MST(P)$, then x is an endpoint of e.

Lemma 1. *Let $x \in \mathbb{R}^2$ be an arbitrary point in the plane, then x belongs to at most 5 lunes in $\mathcal{L}\,(MST(P))$.*

Proof. Let $lune\,(p, q) \in \mathcal{L}\,(MST(P))$ such that $x \in lune\,(p, q)$. We first observe that the edge (p, q) is not in $MST(P \cup \{x\})$. Indeed, recall that each edge in $MST(P \cup \{x\})$ is also an edge in $RNG(P \cup \{x\})$, where $RNG(P \cup \{x\})$ is the relative neighborhood graph of $P \cup \{x\}$, and, since (p, q) is not in $RNG(P \cup \{x\})$, we have that (p, q) is not in $MST(P \cup \{x\})$.

Therefore, by the claim, for each $lune\,(p, q) \in \mathcal{L}\,(MST(P))$ such that $x \in lune\,(p, q)$, $MST(P \cup \{x\})$ contains a unique edge that is connected to x and "replaces" the edge (p, q) of $MST(P)$. However, x is a vertex of an Euclidean minimum spanning tree, and as such its degree is at most 6, implying that x is covered by at most 6 lunes of $\mathcal{L}\,(MST(P))$. In fact, there are at most 5 lunes of $\mathcal{L}\,(MST(P))$ covering x, since one of the edges connected to x (in $MST(P \cup \{x\})$) is present due to the increase in the number of points.

The following theorem summarizes the main result of this section.

Theorem 1. *Let P be a set of points in general position in the plane representing directional antennas of angle $\alpha < 2\pi/3$, and let $s \in P$. One can assign to each $p \in P$ an orientation θ and a range r, such that (i) the resulting $dCG(P)$ contains a path from any $p \in P$ to s, and (ii) $I(dCG(P)) = O(1)$.*

3 Asymmetric Model

In this section we consider the following problem. Assign to each antenna $p \in P$ an orientation θ and a range r, such that $dCG(P)$ is strongly connected and

$I(dCG(P)) = O(1)$. First, we show how to compute such an assignment without bounding the maximum range of the antennas. Then, we show how to do it when the maximum range is bounded by some constant, assuming $UDG(P)$ (i.e., the unit disk graph over P) is connected.

Assuming P is sufficiently large, Bose et al. [3] showed how to select a subset $Q \subseteq P$ of size $l = \lceil 3.5k - 6 \rceil$, where $k = \lceil 2\pi/\alpha \rceil$, and assign to each antenna in Q an orientation θ and range $r = \infty$, such that $dCG(Q)$ contains a tree T rooted at some point (i.e., antenna) $s \in Q$ whose leaves collectively cover the entire plane, and each point in $Q \setminus T$ is oriented towards s. In particular, there exists a path from s to each point of P.

In order to ensure strong connectivity, $dCG(P)$ should also contain, for each $p \in P$, a path from p back to s. First, we modify the orientation assigned to the leaves of T that do not cover any point in $P \setminus T$, by orienting each of them towards s. Notice that this modification does not disrupt any of the existing paths. Next, we handle the points in $P \setminus Q$. In the construction described in [3], the antennas of $P \setminus Q$ are all oriented towards s, but this might induce a graph of interference $\Omega(n)$. We take a different approach and apply the construction described in Section 2. More precisely, we build a converge-cast network for the set $P \setminus Q \cup \{s\}$, where s is the base-station of the network. Notice that the construction of Section 2 does not assign an orientation or range to s, so s's initial orientation and range remain unchanged. Clearly, the interference of the resulting dCG is bounded by the sum of $|Q|$ and the interference of the construction of Section 2 (namely, 5, assuming $\alpha < 2\pi/3$), and is therefore $O(1)$. We thus have the following theorem.

Theorem 2. *Let P be a set of points in general position in the plane representing directional antennas of angle $\alpha < 2\pi/3$. Then, one can orient the antennas and assign ranges to them, such that that the resulting $dCG(P)$ is strongly connected and $I(dCG(P)) = O(1)$.*

The required range of an antenna in the above construction can reach the diameter of P. Assuming $UDG(P)$ is connected, we are able to improve the construction so that the ranges of the antennas are bounded by some constant. Roughly, we "cluster" the points of P and apply the above construction to each of the clusters separately, keeping the ranges of the antennas to within the close vicinity of their cluster. We also make sure that the clusters are connected to each other, while still guaranteeing constant interference. More specifically, we lay a grid $\mathcal{G}$ over P, such that the length of a cell side is $2l$. For a cell $\mathcal{C}$ of $\mathcal{G}$, the *block* of $\mathcal{C}$ is the 3×3 portion of $\mathcal{G}$ centered at $\mathcal{C}$. Each of the 8 cells surrounding cell $\mathcal{C}$ is a *neighbor* of $\mathcal{C}$. A cell of $\mathcal{G}$ is considered *full* if it contains at least l points of P. It is considered *non-full* if it contains at least 1 and at most $l-1$ points of P. Proposition 1 below is analogous to a proposition of Bose et al. [3], referring to a similar grid.

Proposition 1. ([3]) *Let $\mathcal{C}$ be a cell of $\mathcal{G}$. Then, any path in $UDG(P)$ that begins at a point in $\mathcal{C}$ and exits the block of $\mathcal{C}$, must pass through a full cell in $\mathcal{C}$'s block* *(not including $\mathcal{C}$ itself, which may or may not be full). In particular, if there are points of P outside $\mathcal{C}$'s block, then at least one of the 8 neighbors of $\mathcal{C}$ is full.*

In each full cell $\mathcal{C}$, we orient the antennas in $\mathcal{C}$ using the construction described above, with two simple modifications. That is, in each full cell $\mathcal{C}$, we construct the tree $T_{\mathcal{C}}$ rooted at $s_{\mathcal{C}}$, and a converge-cast network for the remaining points in $\mathcal{C}$, where $s_{\mathcal{C}}$ is its base-station. We make the following two modifications: (i) each antenna is assigned range $4\sqrt{2}l$ (instead of unbounded range), and (ii) each leaf $q \in T_{\mathcal{C}}$ is oriented toward $s_{\mathcal{C}}$, unless it either covers a point in $\mathcal{C}$ that is not in $T_{\mathcal{C}}$, or it covers the root of the tree in a full neighboring cell, or it covers a point in a non-full neighboring cell that is oriented towards $s_{\mathcal{C}}$.

In each non-full cell $\mathcal{C}$, we orient each antenna p towards $s_{\mathcal{C}'}$, where $\mathcal{C}'$ is the ($UDG(P)$ hop-distance) closest full cell to p (ties are broken arbitrarily), and assign p a suitable range (which, by definition, does not exceed $4\sqrt{2}l$).

Lemma 2. *The induced dCG is strongly connected.*

Proof. Bose et al. [3] prove that the directed communication graph that is induced by their construction is connected. In their construction, the points in a full cell $\mathcal{C}$ that are not in $T_{\mathcal{C}}$ are oriented towards $s_{\mathcal{C}}$. On the other hand, we construct a converge-cast network for these points, where $s_{\mathcal{C}}$ is its base-station. It is easy to see though that this difference does not affect the strong connectivity property of the induced graph. We thus conclude that the dCG induced from our construction is strongly connected.

The ranges assigned to the antennas do not exceed $4\sqrt{2}l$. Therefore, each $p \in P$ is covered only by antennas lying in the 7×7 block around p's cell. That is, each antenna q can be covered by antennas belonging to a constant number of cells. Since the contribution of all antennas in a single cell to the interference at any point is only $O(1)$, we conclude that the total interference at a point p is $O(1)$.

The following theorem summarizes the main result of this section.

Theorem 3. *Let P be a set of points in general position in the plane representing directional antennas of angle $\alpha < 2\pi/3$, such that $UDG(P)$ is connected. Then, one can orient the antennas and assign ranges to them, such that (i) the induced $dCG(P)$ is strongly connected, (ii) $I(dCG(P)) = O(1)$, and (iii) the maximum range is at most $4\sqrt{2}l$.*

4 Symmetric Model

In this section we consider the problem in the symmetric model. That is, assign to each antenna $p \in P$ an orientation θ and a range r, such that $sCG(P)$ is connected and $I(sCG(P))$ is as small as possible. We first consider the 1D version of the problem, for which we show that it is easy to obtain a connected graph with interference 3. Next, we consider the 2D version. For this version, we first show that there does not always exist a connected graph with low interference. More precisely, for $\alpha < \pi/2$, interference of $\Omega(n)$ is sometimes unavoidable. Our main result for the 2D version is an algorithm for constructing a connected graph with $O(\sqrt{n})$ interference, for $\pi/2 \leq \alpha \leq 3\pi/2$. Assuming UDG is connected, this bound can be replaced by $O(\sqrt{\Delta})$, where Δ is the maximum degree in UDG, and the maximum range of the antennas can be bounded by some constant.

4.1 Interference in 1D

In the *highway model*, where all the antennas lie on the x-axis, one can obtain, for any angle $\alpha < 2\pi$, a connected graph with interference 3, as described below.

Let $p_1 < p_2 < \cdots < p_n$ be n points on the x-axis. For simplicity assume that $n \geq 6$ and that n is even. Orient each point p_i in odd position to the right and assign to it a range that is just sufficient to reach p_{i+3}, for $1 \leq i \leq n-3$, or p_n, for $i = n-1$. Symmetrically, orient each point p_i in even position to the left and assign to it a range that is just sufficient to reach p_{i-3}, for $4 \leq i \leq n$, or p_1 for $i = 2$. See Figure 3 for an illustration. It is easy to see that the induced communication graph is connected and that each point lies in the range of at most three other points.

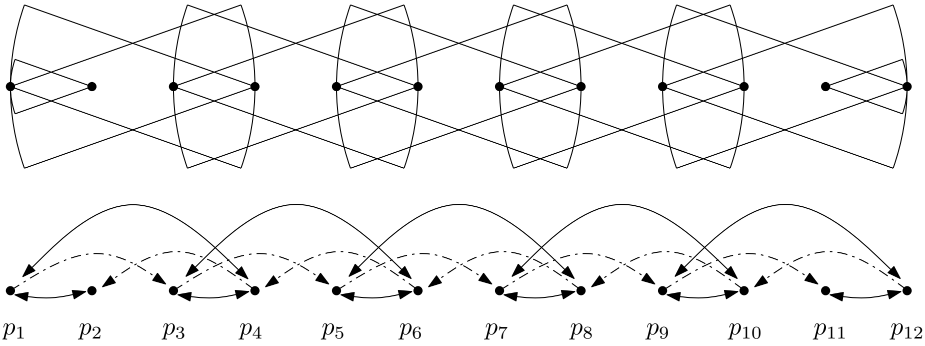


Fig. 3. Two representations of a connected graph with interference 3 for points on the x-axis and $\alpha < 2\pi$. In the bottom representation, an arrow from p_i to p_j indicates that p_i covers p_j. The bidirectional arrows are the edges of the graph.

Lemma 3. *Let P be a set of n points on the x-axis representing directional antennas of angle α, where $\alpha < 2\pi$. Then, one can orient the antennas and assign ranges to them, such that the resulting $sCG(P)$ is connected and $I(sCG(P)) = 3$.*

Remark. It is easy to verify that already for 5 points, there does not exist a connected graph with interference less than 3, thus, 3 is optimal.

4.2 Interference in 2D

$\boldsymbol{\alpha < \pi/2}$. Recall that if $\alpha < \pi/3$, there does not always exist a connected graph. We show that for $\pi/3 \leq \alpha < \pi/2$ there does not always exist a connected low-interference graph. Consider, for example, the following setting. The points

$p_1, \ldots, p_{n-1}$ are located on some vertical line and the point p_n is located far enough to the right, so that it is impossible for p_i, $i = 1, \ldots, n-1$, to cover both p_n and some other point p_j, $1 \leq j \leq n-1$. The only way to ensure connectivity in this case is to orient $p_1, \ldots, p_{n-1}$ towards p_n, but then $I(sCG(P)) = I(p_n) = n-1$.

$\pi/2 \leq \alpha \leq 3\pi/2$. For larger angles, the situation is much better. Given n directional antennas of angle $\pi/2 \leq \alpha \leq 3\pi/2$, we are able to construct a connected graph with interference $O(\sqrt{n})$. We first observe that the construction described in Section 4.1 for the case of points on the x-axis, can also be used for the case of an xy-monotone sequence of points. That is, consider the sequence s obtained by sorting the input points by their x-coordinate. If s happens to be monotone (either increasing or decreasing), then two orientations are sufficient in order to connect the points in s, and, moreover, one can do so such that the resulting interference is 3. More specifically, instead of orientations 0 and π (i.e., right and left) that are used to connect points on the x-axis, we use orientations $\pi/4$ and $5\pi/4$, if s is increasing, and orientations $3\pi/4$ and $7\pi/4$, if s is decreasing, in order to connect the points in s. Notice that after connecting the points in s, the interference of any point in the plane is at most 4.

We thus start by sorting the points of P by their x-coordinate. By the well-known theorem of Erdös and Szekeres [9], the resulting sorted sequence can be partitioned into $m = O(\sqrt{n})$ xy-monotone subsequences. (This can be done easily by repeatedly finding the longest monotone subsequence.) We could consider each sequence s of the m sequences separately, and connect the points in s as described above. This would ensure that the interference at any point in the plane is at most $4m = O(\sqrt{n})$. However, the sequences might not be connected to each other.

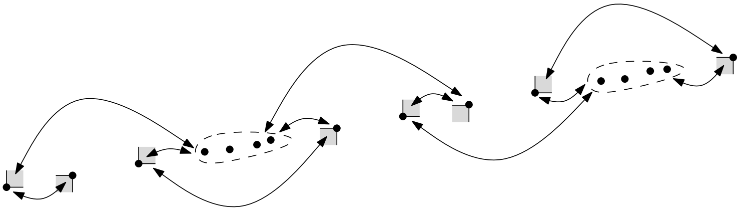


Fig. 4. A monotone sequence. The two quadruplets are marked by dashed balloons. The sequence is connected and its interference is at most 11.

We describe how to overcome this problem. Consider a sequence $s = (p_1, \ldots, p_l)$ and assume that s is long, i.e., $l \geq 8$. Arbitrarily choose two disjoint quadruplets of points in s, where a quadruplet consists of four consecutive points. For each of these quadruplets Q, assign orientations and ranges to the points in Q, so that Q is connected and covers all the points in $P \setminus Q$. This is possible by a result of Aschner et al. [2]. Now, assign orientations and ranges to the remaining points in s, as described above, referring to each of the quadruplets as a pair of antennas (see Figure 4). Notice that since each quadruplet Q is connected and covers the two points of s that need to connect to it, s is connected and the interference of s is at most 11 (actually, 7 by a more delicate argument). Consequently, the interference at any point in the plane is at most $12m = O(\sqrt{n})$.

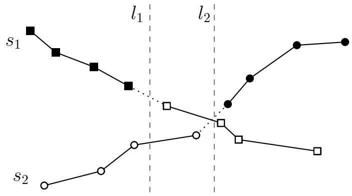


Fig. 5. The quadruplet of s_1 to the left of l_1 and the quadruplet of s_2 to the right of l_2 can be separated by a line, implying that s_1 and s_2 are connected.

Consider any two long sequences s_1 and s_2. We claim that s_1 and s_2 are connected. Aschner et al. [2] showed that any two quadruplets that can be separated by a line are connected. Let l_1 (resp., l_2) be a vertical line separating between s_1's (resp. s_2's) two quadruplets, and assume w.l.o.g. that l_1 is to the left of l_2. Then, clearly the quadruplet of s_1 to the left of l_1 and the quadruplet of s_2 to the right of l_2 can be separated by a line, implying that s_1 and s_2 are connected; see Figure 5.

Finally, we describe how to handle the short sequences. For each short sequence s and for each point p in s, simply pick an arbitrary point q that covers p and orient p towards q and set p's range accordingly. Such a point q always exists, since in each quadruplet there exists such a point. This increases the interference by only a constant, since it is easy to ensure that all but a constant number of sequences are long.

The following theorem summarizes our last result.

Theorem 4. *Let P be a set of points in the plane representing directional antennas of angle $\pi/2 \leq \alpha \leq 3\pi/2$. Then, one can orient the antennas and assign ranges to them, such that the induced $sCG(P)$ is strongly connected and $I(sCG(P)) = O(\sqrt{n})$.*

Bounding the range and refining the bound on the interference. In the above construction we did not make any effort to reduce the ranges assigned to the antennas. Assuming $UDG(P)$ is connected, we now show how to bound the ranges by some constant and reduce the interference to $O(\sqrt{\Delta})$, where Δ is the maximum degree in $UDG(P)$. Lay a grid $\mathcal{G}$ over P, such that the length of a cell side is 7. A cell of $\mathcal{G}$ is considered *full* if it contains at least 4 points of P. It is considered *non-full* if it contains at least 1 and at most 3 points of P. Notice that the number of points of P in any $\frac{1}{2} \times \frac{1}{2}$ square is at most $\Delta + 1$, since these points correspond to a clique in $UDG(P)$. Therefore, each cell in $\mathcal{G}$ contains only $O(\Delta)$ points.

Roughly, we apply the construction described above in each cell $\mathcal{C}$ independently, and restrict the ranges of the points in a quadruplet so that they are just enough to reach any point in $\mathcal{C}$ or in one of its eight neighbor cells. Notice that if a cell $\mathcal{C}$ contains at least 64 points, then one of the monotone sequences is of length at least 8, and therefore, the neighbor cells of $\mathcal{C}$ are covered by the two quadruplets of that sequence. If a cell $\mathcal{C}$ is full but contains less than 64 points, then we choose four arbitrary "hub" points to cover $\mathcal{C}$ and its neighbor cells, and orient each of the remaining points in $\mathcal{C}$ to an arbitrary hub point that covers it. This ensures that each full cell $\mathcal{C}$ contains at least one quadruplet that covers $\mathcal{C}$ and its eight neighbor cells, and therefore, that any two full cells that are neighbors of each other are connected (since they can be separated by a line, see [2]). If a cell $\mathcal{C}$ is non-full, then we orient each point in $\mathcal{C}$ to a point that covers it in a neighboring full cell. By Proposition 1 it is not hard to see that the resulting graph is connected.

Finally, we claim that the interference is $O(\sqrt{\Delta})$. This is true since the interference caused by each cell is $O(\sqrt{\Delta})$, and any point in the plane is under the influence of only a constant number of cells surrounding it. Therefore the interference of the final graph is $O(\sqrt{\Delta})$. The following theorem summarizes the main results of this section.

Theorem 5. *Let P be a set of points in the plane representing directional antennas, such that $UDG(P)$ is connected. Then, (i) for $\pi/2 \leq \alpha \leq 3\pi/2$, one can orient the antennas and assign ranges to them, such that the induced $sCG(P)$ is strongly connected, $I(sCG(P)) = O(\sqrt{\Delta})$, where Δ is the maximum degree in UDG, and the maximum range is at most $O(1)$, and (ii) for $\alpha < \pi/2$, $I(sCG) = \Omega(n)$ is sometimes unavoidable.*

5 Conclusion and Future Work

In this paper we tackled the question raised in the title. Do directional antennas facilitate in reducing interferences, as commonly believed. In the asymmetric model, the answer is positive. Directional antennas of angle $\alpha < 2\pi/3$ enable reduction in the interference from $\Theta(\log \Delta)$ to $O(1)$. Although, from a theoretical point of view, it might be interesting to investigate the case of angles larger than $2\pi/3$, we believe that it is less interesting from a practical point of view, since, as the angle increases, the advantages in using directional antennas (e.g., energy efficiency) diminish. In the symmetric model, directional antennas might be bad in the context of interference. For small angles (less than $\pi/2$), not only that the interference does not decrease w.r.t. the $\Theta(\sqrt{\Delta})$-bound, it even increases to $\Theta(n)$. This is rather surprising, since intuitively the interference should decrease as the angle decreases. For larger angles, i.e., for $\pi/2 \leq \alpha \leq 3\pi/4$, we have shown how to obtain interference $O(\sqrt{\Delta})$, thus matching the upper bound for omnidirectional antennas. A possible explanation for these counterintuitive results is that when using directional antennas it is more difficult to achieve connectivity and longer ranges might be required for that. Finally, we note that for $\alpha = 2\pi - \varepsilon$, where $\varepsilon = O(1/2^n)$, a lower bound of $\Omega(\sqrt{\Delta})$ can be easily derived from an example obtained from the exponential chain example by slightly changing it so that no three points are collinear. The main open problem is either to improve the $O(\sqrt{\Delta})$-bound in the symmetric model, or to prove a matching lower bound.

References

1. Ackerman, E., Gelander, T., Pinchasi, R.: Ice-creams and wedge graphs. CoRR abs/1106.0855 (2011)
2. Aschner, R., Katz, M.J., Morgenstern, G.: Symmetric connectivity with directional antennas. CoRR abs/1108.0492 (2011)
3. Bose, P., Carmi, P., Damian, M., Flatland, R., Katz, M.J., Maheshwari, A.: Switching to Directional Antennas with Constant Increase in Radius and Hop Distance. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 134–146. Springer, Heidelberg (2011)
4. Burkhart, M., von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Does topology control reduce interference? In: Proc. 5th ACM Internat. Sympos. on Mobile Ad Hoc Networking and Computing, pp. 9–19 (2004)
5. Caragiannis, I., Kaklamanis, C., Kranakis, E., Krizanc, D., Wiese, A.: Communication in wireless networks with directional antennas. In: 20th ACM Sympos. on Parallelism in Algorithms and Architectures, pp. 344–351 (2008)
6. Carmi, P., Katz, M.J., Lotker, Z., Rosén, A.: Connectivity guarantees for wireless networks with directional antennas. Computational Geometry: Theory and Applications 44(9), 477–485 (2011)
7. Carmi, P., Katz, M.J., Mitchell, J.S.B.: The minimum-area spanning tree problem. Computational Geometry: Theory and Applications 35(3), 218–225 (2006)
8. Damian, M., Flatland, R.: Spanning properties of graphs induced by directional antennas. In: Electronic Proc. 20th Fall Workshop on Computational Geometry. Stony Brook, NY (2010)
9. Erdös, P., Szekeres, G.: A combinatorial problem in geometry. Compositio Mathematica 2, 463–470 (1935)
10. Halldórsson, M.M., Tokuyama, T.: Minimizing interference of a wireless ad-hoc network in a plane. Theor. Comput. Sci. 402(1), 29–42 (2008)
11. Johansson, T., Carr-Motycková, L.: Reducing interference in ad hoc networks through topology control. In: Proc. of the 2005 Joint Workshop on Foundations of Mobile Computing, pp. 17–23 (2005)
12. Korman, M.: Minimizing Interference in Ad-Hoc Networks with Bounded Communication Radius. In: Asano, T., Nakano, S.-i., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 80–89. Springer, Heidelberg (2011)
13. Kranakis, E., Krizanc, D., Morales, O.: Maintaining connectivity in sensor networks using directional antennae. In: Nikoletseas, S., Rolim, J.D.P. (eds.) Theoretical Aspects of Distributed Computing in Sensor Networks, ch. 3, pp. 59–84. Springer (2011)
14. Moaveni-Nejad, K., Li, X.-Y.: Low-interference topology control for wireless ad hoc networks. Ad Hoc & Sensor Wireless Networks 1(1-2) (2005)
15. von Rickenbach, P., Schmid, S., Wattenhofer, R., Zollinger, A.: A robust interference model for wireless ad-hoc networks. In: Proc. 19th IEEE Internat. Parallel and Distributed Processing Sympos. (2005)
16. Tan, H., Lou, T., Lau, F.C.M., Wang, Y., Chen, S.: Minimizing Interference for the Highway Model in Wireless Ad-Hoc and Sensor Networks. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Královič, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 520–532. Springer, Heidelberg (2011)

Minimum Convex Partitions and Maximum Empty Polytopes*

Adrian Dumitrescu[1], Sariel Har-Peled[2], and Csaba D. Tóth[3]

[1] Computer Science, University of Wisconsin–Milwaukee
dumitres@uwm.edu
[2] Computer Science, University of Illinois at Urbana–Champaign
sariel@cs.uiuc.edu
[3] Mathematics and Statistics, Univ. of Calgary and Comp. Sci., Tufts University
cdtoth@ucalgary.ca

Abstract. Let S be a set of n points in d-space. A convex Steiner partition is a tiling of $\text{conv}(S)$ with empty convex bodies. For every integer d, we show that S admits a convex Steiner partition with at most $\lceil (n-1)/d \rceil$ tiles. This bound is the best possible for points in general position in the plane, and it is best possible apart from constant factors in every fixed dimension $d \geq 3$. We also give the first constant-factor approximation algorithm for computing a minimum Steiner convex partition of a planar point set in general position.

Establishing a tight lower bound for the maximum volume of a tile in a Steiner partition of any n points in the unit cube is equivalent to a famous problem of Danzer and Rogers. It is conjectured that the volume of the largest tile is $\omega(1/n)$ in any dimension $d \geq 2$. Here we give a $(1-\epsilon)$-approximation algorithm for computing the maximum volume of an empty convex body amidst n given points in the d-dimensional unit box $[0,1]^d$.

1 Introduction

Let S be a set of $n \geq d+1$ points in $\mathbb{R}^d$, $d \geq 2$. A convex body C is *empty* if its interior is disjoint from S. A *convex partition* of S is a partition of the convex hull $\text{conv}(S)$ into empty convex bodies (called *tiles*) such that the vertices of the tiles are in S. In a *convex Steiner partition* of S the vertices of the tiles are arbitrary: they can be points in S or new *Steiner points*. For instance, any triangulation of S is a convex partitions of S, where the convex bodies are simplices, and so $\text{conv}(S)$ can always be partitioned into fewer than dn empty convex tiles.

In this paper, we study the minimum number of tiles that a convex Steiner partition of every n points in $\mathbb{R}^d$ admits, and the maximum volume of a single tile for a given point set. The research is motivated by a longstanding open problem by Danzer and Rogers [2,3,4,11]: What is the maximum volume of an

* Dumitrescu is supported in part by the NSF grant DMS-1001667; Har-Peled is supported in part by the NSF grant CCF-0915984; Tóth acknowledges support from the NSERC grant RGPIN 35586.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 213–224, 2012.

empty convex body $C \subset [0,1]^d$ that can be found amidst any set $S \subset [0,1]^d$ of n points in a unit cube? The current best bounds are $\Omega(1/n)$ and $O(\log n/n)$, respectively (for a fixed d). The lower bound comes from decomposing the unit cube by n parallel hyperplanes, each containing at least one point, into at most $n+1$ empty convex bodies. The upper bound is tight apart from constant factors for n uniformly distributed random points in the unit cube. It is suspected that the largest volume (as a function of n) grows faster then $\Omega(1/n)$, i.e., it is $\omega(1/n)$ in any dimension $d \geq 2$.

Minimum Number of Tiles in a Convex Partition. A *minimum convex partition* of S is a convex partition of S with a minimum number of tiles. Denote this number by $f_d(S)$. Further define (by slightly abusing notation)

$$f_d(n) = \max\{f_d(S) : S \subset \mathbb{R}^d, |S| = n\}.$$

Similarly define a *minimum Steiner convex partition* of S as one with a minimum number of tiles and let $g_d(S)$ denote this number. We also define

$$g_d(n) = \max\{g_d(S) : S \subset \mathbb{R}^d, |S| = n\}.$$

There has been substantial work on estimating $f_2(n)$, and computing $f_2(S)$ for a given set S in the plane. It has been shown successively that $f_2(n) \leq \frac{10n-18}{7}$ by Neumann-Lara et al. [20], $f_2(n) \leq \frac{15n-24}{11}$ by Knauer and Spillner [17], and $f_2(n) \leq \frac{4n-6}{3}$ for $n \geq 6$ by Sakai and Urrutia [21]. From the other direction, García López and Nicolás [12] proved that $f_2(n) \geq \frac{12n-22}{11}$, for $n \geq 4$, thereby improving an earlier lower bound $f_2(n) \geq n+2$ by Aichholzer and Krasser [1]. Knauer and Spillner [17] have also obtained a $\frac{30}{11}$-factor approximation algorithm for computing a minimum convex partition for a given set $S \subset \mathbb{R}^2$, no three of which are collinear. There are also a few exact algorithms, including three fixed-parameter algorithms [10,13,22].

The state of affairs is much different in regard to convex Steiner partitions. As pointed out in [8], no corresponding results are known for the variant with Steiner points. Here we take the first steps in this direction, and obtain the following results.

Theorem 1. *For $n \geq d+1$, we have $g_d(n) \leq \left\lceil \frac{n-1}{d} \right\rceil$. For $d = 2$, this bound is the best possible, that is, $g_2(n) = \lceil (n-1)/2 \rceil$; and for every fixed $d \geq 2$, we have $g_d(n) \geq \Omega(n)$.*

We say that a set of points in $\mathbb{R}^d$ is in *general position* if every k-dimensional affine subspace contains at most $k+1$ points for $0 \leq k < d$. We show that in the plane every convex Steiner partition for n points in general position, i of which lie in the interior of the convex hull, has at least $\Omega(i)$ tiles. This leads to a simple constant-factor approximation algorithm.

Theorem 2. *Given a set S of n points in general position in the plane, a ratio 3 approximation of a minimum convex Steiner partition of S can be computed in $O(n \log n)$ time.*

The *average* volume of a tile in a Steiner partition of n points in the unit cube $[0,1]^d$ is an obvious lower bound for the maximum possible volume of a tile, and for the maximum volume of any empty convex body $C \subset [0,1]^d$. The lower bound $g_d(n) \geq \Omega(n)$ in Theorem 1 shows that the average volume of a tile is $O(1/n)$ in some instances, where the constant of proportionality depends only on d. This implies that a simple "averaging" argument is not a viable avenue for finding a solution to the problem of Danzer and Rogers.

Maximum Empty Polytope Among n Points in a Unit Cube. In the second part of the paper, we consider the following problem: Given a set of n points in rectangular box B in $\mathbb{R}^d$, find a maximum-volume empty convex body $C \subset B$. Since the ratio between volumes is invariant under affine transformations, we may assume without loss of generality that $B = [0,1]^d$. We therefore have the problem of computing a maximum volume empty convex body $C \subset [0,1]^d$ for a set of n points in $[0,1]^d$. We are not aware of any exact algorithm even in the plane. It can be argued that the maximum volume empty convex body is a polytope, however, the (number and) location of its vertices is unknown and this represents the main difficulty.

By John's ellipsoid theorem, the maximum volume empty ellipsoid in $[0,1]^d$ gives a $\frac{1}{d^d}$-approximation. Here we present a $(1-\varepsilon)$-approximation for the maximum volume empty convex body C_{opt} by first guessing an approximate inscribed ellipsoid of C_{opt}, which is an empty ellipsoid whose volume is an $\frac{1}{d^d}$-approximation of $\text{vol}(C_{\text{opt}})$, and then refining it to a sufficiently close approximation of C_{opt}.

Theorem 3. *Given a set S of n points in $[0,1]^d$, one can $(1-\varepsilon)$-approximate the maximum-volume empty convex body in $[0,1]^d$. The running time of the approximation algorithm is*

$$O\Big(n^{1+d(d-1)/2} 2^{O\left(1/\varepsilon^d\right)} \log^d n\Big).$$

As far as the problem of Danzer and Rogers is concerned, say in the plane, one need not consider convex sets—it suffices to consider triangles—and for triangles the problems considered are much simpler. That is, the asymptotic dependency on n of the areas of the largest empty triangle and convex body are the same.

2 Combinatorial Bounds

In this section we prove Theorem 1. We start with the upper bound. The following simple algorithm returns a convex Steiner partition with at most $\lceil (n-1)/d \rceil$ tiles for any n points in $\mathbb{R}^d$.

Algorithm **A1**:

```
Step 1. Compute the convex hull $R \leftarrow \text{conv}(S)$ of $S$. Let $A \subseteq S$ be the set of
    hull vertices, and let $B = S \setminus A$ denote the remaining points.
```

Step 2. Compute $\text{conv}(B)$, and let H be the supporting hyperplane of an arbitrary $(d-1)$-dimensional face of $\text{conv}(B)$. Denote by H^+ the halfspace that contains B, and $H^- = \mathbb{R}^d \setminus H^+$. The hyperplane H contains d points of B, and it decomposes R into two convex bodies: $R \cap H^-$ is empty and $R \leftarrow R \cap H^+$ contains all points in $B \setminus H$. Update $B \leftarrow B \setminus H$ and $R \leftarrow R \cap H^+$.

Step 3. Repeat Step 2 with the new values of R and B until B is the empty set. (If $|B| < d$, then any supporting hyperplane of B completes the partition.)

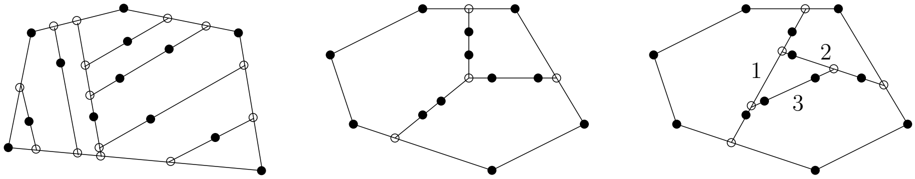


Fig. 1. Steiner convex partitions with Steiner points drawn as hollow circles. Left: A convex Steiner partition of a set of 13 points. Middle: A Steiner partition of a set of 12 points into three tiles. Right: A Steiner partition of the same set of 12 points into 4 tiles, generated by Algorithm **A1** (the labels reflect the order of execution).

It is obvious that the algorithm generates a Steiner convex partition of S. An illustration of Algorithm **A1** on a small planar example appears in Figure 1(right). Let h and i denote the number of hull and interior points of S, respectively, so that $n = h + i$. Each hyperplane used by the algorithm removes d interior points of S (with the possible exception of the last round if i is not a multiple of d). Hence the number of convex tiles is $1 + \lceil i/d \rceil$, and we have $1 + \lceil i/d \rceil = \lceil (i+d)/d \rceil \leq \lceil (n-1)/d \rceil$, as required.

Lower Bound in the Plane. A matching lower bound in the plane is given by the following construction. For $n \geq 3$, let $S = A \cup B$, where A is a set of 3 noncollinear points in the plane, and B is a set of $n-3$ points that form a regular $(n-3)$-gon in the interior of $\text{conv}(A)$, so that $\text{conv}(S) = \text{conv}(A)$ is a triangle. If $n = 3$, then $\text{conv}(S)$ is an empty triangle, and $g_2(S) = 1$. If $4 \leq n \leq 5$, S is not in convex position, and so $g_2(S) \geq 2$. Suppose now that $n \geq 6$.

Consider an arbitrary convex partition of S. Let o be a point in the interior of $\text{conv}(B)$ such that the lines os, $s \in S$, do not contain any edges of the tiles. Refer to Figure 2(left). For each point $s \in B$, choose a *reference point* $r(s) \in \mathbb{R}^2$ on the ray $\overrightarrow{os}$ in $\text{conv}(A) \setminus \text{conv}(B)$ sufficiently close to point s, and lying in the interior of a tile. Note that the convex tile containing o cannot contain any reference points. We claim that any tile contains at most 2 reference points. This immediately implies $g_2(S) \geq 1 + \lceil (n-3)/2 \rceil = \lceil (n-1)/2 \rceil$.

Suppose, to the contrary, that a tile τ contains 3 reference points r_1, r_2, r_3, corresponding to the points s_1, s_2, s_3. Refer to Figure 2. Note that o cannot be in the interior of τ, otherwise τ would contain all points s_1, s_2, s_3 in its interior. Hence $\text{conv}\{o, s_1, s_2, s_3\}$ is a quadrilateral, and $\text{conv}\{o, r_1, r_2, r_3\}$ is also a

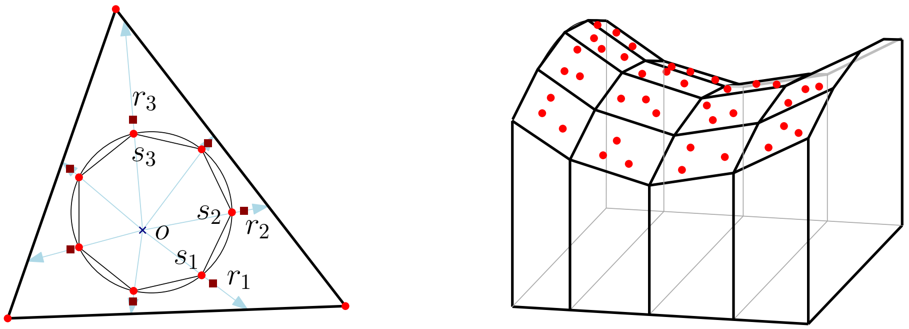


Fig. 2. Left: Lower bound construction in $\mathbb{R}^2$. Right: Points in general position on a saddle surface in $\mathbb{R}^3$.

quadrilateral, since the reference points are sufficiently close to the corresponding points in B. We may assume w.l.o.g. that vertices of $\text{conv}\{o, s_1, s_2, s_3\}$ are o, s_1, s_2, s_3 in counterclockwise order. Then s_2 lies in the interior of $\text{conv}\{o, r_1, r_2, r_3\}$. We conclude that every tile τ contains at most 2 reference points, as required.

Lower bounds for $d \geq 3$. A similar construction works in for any $d \geq 2$, but the lower bound no longer matches the upper bound $g_d(n) \leq \lceil (n-1)/d \rceil$ for $d \geq 3$.

Recall that a *Horton set* [16] is a set S of n points in the plane such that the convex hull of any 7 points is non-empty. Valtr [23] generalized Horton sets to d-space. For every $d \in \mathbb{N}$, there exists a minimal integer $h(d)$ with the property that for every $n \in \mathbb{N}$ there is a set S of n points in general position in $\mathbb{R}^d$ such that the convex hull of any $h(d) + 1$ points in S is non-empty. It is known that $h(2) = 6$, and Valtr proved that $h(3) \leq 22$, and in general that $h(d) \leq 2^{d-1}(N(d-1)+1)$, where $N(k)$ is the product of the first k primes.

We construct a set S of $n \geq d+1$ points in $\mathbb{R}^d$ as follows. Let $S = A \cup B$, where A is a set of $d+1$ points in general position in $\mathbb{R}^d$, and B is a generalized Horton set of $n-(d+1)$ points in the interior of $\text{conv}(A)$, such that the interior of any $h(d) + 1$ points from B contains some point in B.

Consider an arbitrary convex Steiner partition of S. Every point $b \in B$ is in the interior of $\text{conv}(S)$, and so it lies on the boundary of at least 2 convex tiles. For each $b \in B$, place two *reference points* in the interiors of 2 distinct tiles incident to b. Every tile contains at most $h(d)$ reference points. Indeed, if a tile contains $h(d) + 1$ reference points, then it is incident to $h(d) + 1$ points in B, and some point of B lies in the interior of the convex hull of these points, a contradiction.

We have $2(n-d-1)$ reference points, and every tile contains at most $h(d)$ of them. So the number of tiles is at least $\lceil 2(n-d-1)/h(d) \rceil$.

3 Approximating the Minimum Convex Steiner Partition in $\mathbb{R}^2$

In this section we prove Theorem 2 by showing that our simple-minded algorithm **A1** from Section 2 achieves a constant-factor approximation in the plane if the points in S are in general position.

Approximation Ratio. Recall that algorithm **A1** computes a Steiner partition of $\mathrm{conv}(S)$ into at most $1+\lceil i/2\rceil$ parts, where i stands for the number of interior points of S. If $i=0$, the algorithm computes an optimal partition, i.e., $\mathrm{ALG} = \mathrm{OPT} = 1$. Assume now that $i \geq 1$. Consider an optimal convex Steiner partition Π of S with OPT tiles. We construct a planar multigraph $G=(V,E)$ as follows. The *faces* of G are the convex tiles and the exterior of $\mathrm{conv}(S)$ (the outer face). The *vertices* V are the points in the plane incident to at least 3 faces (counting the outer face as well). Since $i \geq 1$, G is non-empty and we have $|V| \geq 2$. Each *edge* in E is a Jordan arc on the common boundary of two faces. An edge between two bounded faces is a straight line segment, and so it contains at most two interior points of S. An edge between the outer face and a bounded face is a convex arc, containing hull points from S. Double edges are possible if two vertices of the outer face are connected by a straight line edge and a curve edge along the boundary—in this case these two parallel edges bound a convex face. No loops are possible in G. Since Π is a convex partition, G is connected.

Let v, e, and f, respectively, denote the number of vertices, edges, and bounded (convex) faces of G; in particular, $f=\mathrm{OPT}$. By Euler's formula for planar multigraphs, we have $v-e+f=1$, that is, $f=e-v+1$. By construction, each vertex of G is incident to at least 3 edges, and every edge is incident to two vertices. Therefore, $3v \leq 2e$, or $v \leq 2e/3$. Consequently, $f=e-v+1 \geq e-2e/3+1=e/3+1$. Since S is in general position, each straight-line edge of G contains at most 2 interior points from S. Curve edges along the boundary do not contain interior points. Hence each edge in E is incident to at most two interior points in S, thus $i \leq 2e$. Substituting this into the previous inequality on f yields $\mathrm{OPT}=f \geq e/3+1 \geq i/6+1$. Comparing this lower bound with the upper bound $\mathrm{ALG} \leq \lceil i/2\rceil + 1$, we conclude that

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} \leq \frac{\lceil i/2\rceil+1}{i/6+1} \leq 3\,\frac{i+3}{i+6} < 3,$$

and the approximation ratio of 3 follows.

Tightness of the Approximation Ratio. We first show that the above ratio 3 is tight for Algorithm **A1**. We construct a planar point set S as follows. Consider a large (say, hexagonal) section of a hexagonal lattice. Place Steiner vertices at the lattice points, and place two points in S on each lattice edge. Slightly perturb the lattice, and add a few more points in S near the boundary, and a few more Steiner points, so as to obtain a convex Steiner partition of S with no three points collinear. Denote by v, e, and f, the elements of the planar multigraph

G as before. Since we consider a large lattice section, we have $v, e, f \to \infty$. We write $a \sim b$, whenever $a/b \to 1$. As before, we have $f + v = e + 1$, and since each non-boundary edge is shared by two convex faces, we have $e \sim 6f/2 = 3f$. By construction, $i \sim 2e \sim 6f$, hence $f \sim i/6$. Therefore the convex partition constructed above has $f \sim i/6$, while Algorithm **A1** constructs one with about $i/2$ faces. Letting $e \to \infty$, then $i \to \infty$, and the ratio ALG/OPT approaches 3 in the limit: ALG/OPT $\sim (i/2)/(i/6) = 3$.

Time Analysis. It is easy to show that Algorithm **A1** runs in $O(n \log n)$ time for a set S of n points in the plane. We employ the semi-dynamic (delete only) convex hull data structure of Hershberger and Suri [15]. This data structure supports point deletion in $O(\log n)$ time, and uses $O(n)$ space and $O(n \log n)$ preprocessing time. We maintain the boundary of a convex polygon R in a binary search tree, a set $B \subset S$ of points lying in the interior of R, and the convex hull $\mathrm{conv}(B)$ with the above semi-dynamic data structure [15]. Initially, $R = \mathrm{conv}(S)$, which can be computed in $O(n \log n)$ time; and $B \subset S$ is the set of interior points. In each round of the algorithm, consider the supporting line H of an arbitrary edge e of $\mathrm{conv}(B)$ such that B lies in the halfplane H^+. The two intersection points of H with the boundary of R can be computed in $O(\log n)$ time. At the end of the round, we can update $B \leftarrow B \setminus H$ and $\mathrm{conv}(B)$ in $O(k \log n)$ time, where k is the number of points removed from B; and we can update $R \leftarrow R \cap H^+$ in $O(\log n)$ time. Every point is removed from B exactly once, and the number of rounds is at most $\lceil (n-3)/2 \rceil$, so the total update time is $O(n \log n)$ throughout the algorithm.

Remark. Interestingly enough, in dimensions 3 and higher, Algorithm **A1** does not give a constant-factor approximation. For every integer n, one can construct a set S of n points in general position in $\mathbb{R}^3$ such that $i = n - 4$ of them lie in the interior of $\mathrm{conv}(S)$, but the minimum convex Steiner partition has only $O(\sqrt{n})$ tiles. In contrast, Algorithm **A1** computes a Steiner partition with $i/3 = (n-4)/3$ convex tiles.

We first construct the convex tiles, and then describe the point set S. Specifically, S consists of 4 points of a large tetrahedron, and 3 points in general position on the common boundary of certain pairs of adjacent tiles.

Let $k = \lceil \sqrt{(n-4)/3} \rceil$. Place $(k+1)^2$ Steiner points $(a, b, a^2 - b^2)$ on the saddle surface $z = x^2 - y^2$ for pairs of integers $(a, b) \in \mathbb{Z}^2$, $-\lfloor k/2 \rfloor \le a, b \le \lceil k/2 \rceil$. The four points $\{(x, y, x^2 - y^2) : x \in \{a, a+1\}, y \in \{b, b+1\}\}$ form a parallelogram for every $(a, b) \in \mathbb{Z}^2$, $-\lfloor k/2 \rfloor \le a, b \le \lceil k/2 \rceil - 1$. Refer to Figure 2(right). These parallelograms form a terrain over the region $\{(x, y) : -\lfloor k/2 \rfloor \le x, y \le \lceil k/2 \rceil\}$. Note that no two parallelograms are coplanar. Subdivide the space *below* this terrain by vertical planes $x = a$, $-\lfloor k/2 \rfloor \le a \le \lceil k/2 \rceil$. Similarly, subdivide the space *above* this terrain by planes $y = b$, $-\lfloor k/2 \rfloor \le b \le \lceil k/2 \rceil$. We obtain $2k$ interior-disjoint convex regions, k above and k below the terrain, such that the common boundary of a region above and a region below is a parallelogram of the terrain. The points in $\mathbb{R}^3$ that do not lie above or below the terrain can be covered by 4 convex wedges.

Enclose the terrain in a sufficiently large tetrahedron T. Clip the $2k$ convex regions and the 4 wedges into the interior of T. These $2k + 4$ convex bodies tile T. Choose 3 non-collinear points of S in each of the k^2 parallelograms, such that no 4 points are coplanar and no 2 are collinear with vertices of T. Let the point set S be the set of 4 vertices of the large tetrahedron T and the $3k^2$ points selected from the parallelograms.

4 Approximating the Maximum Empty Convex Body

Let S be a set of points in the unit cube $[0, 1]^d \subseteq \mathbb{R}^d$. Our task is to approximate the largest convex body $C \subseteq [0, 1]^d$ that contains no points of S in its interior. Let $C_{\text{opt}} = C_{\text{opt}}(S)$ denote this body, and let $\text{vol}_{\text{opt}}(S)$ denote its volume.

As mentioned in the Introduction (see also Theorem 1), $\text{vol}(C_{\text{opt}}) \geq \Omega(1/n)$. The diameter of C_{opt} is bounded from above by $\sqrt{d}$, and its width is bounded from below by $1/(n + 1)$. By John's ellipsoid theorem [19], for any compact convex body C in $\mathbb{R}^d$ there exists an ellipsoid $\mathcal{E}$ such that $\mathcal{E} \subseteq C \subseteq d\mathcal{E}$ where we denote by $k\mathcal{E}$ the ellipsoid $\mathcal{E}$ scaled up by a factor of k and having the same center as $\mathcal{E}$. It follows that $\text{vol}(C) / d^d \leq \text{vol}(\mathcal{E}) \leq \text{vol}(C)$. We need the following well-known fact; see [7] for a proof.

Lemma 1. *Every ellipsoid in $\mathbb{R}^d$ contains at most $d^2 + d$ points in general position.*

Lemma 2. *Assume that $\mathcal{E} \subseteq [0, 1]^d$ is an ellipsoid, and $\text{vol}(\mathcal{E}) \geq \rho$. Then, one can compute a set Q of $O(1/\rho \log(1/\rho))$ points such that, with probability $\geq 1 - (\rho/2)^{O(1)}$, one of the points of Q lies in $\mathcal{E}/2$.*

Proof. Observe that ellipsoids in $\mathbb{R}^d$ have bounded VC dimension (see [18] for basic definitions). Indeed, by Lemma 1 an ellipsoid has at most $d^2 + d$ points of S on its boundary. Now, given an ellipsoid we transform it continuously into an equivalent ellipsoid containing the same set of points, while having a maximum number of points on its boundary. As such, its shattering dimension is bounded by $O(d^2)$, thus its VC dimension is bounded by $d' = O(d^2 \log d)$. (It is likely that a better bound on the VC dimension is possible by being more careful.)

By the ε-net theorem [18, Ch. 5.2], a sample Q of size $O(d'/\rho \log 1/\rho)$ will hit any ellipsoid in the unit cube with volume $\geq \rho/2^d$ with high probability. In particular, this sample hits $\mathcal{E}/2$. □

4.1 Handling the Large Volume Case

In the following, assume that $m > 0$ is some integer, and consider the grid point set $\mathcal{G}(m) = \left\{ (i_1, \ldots, i_d)/m \;\middle|\; i_1, \ldots, i_d \in \{0, 1, \ldots, m\} \right\}$. Let $S \subseteq [0, 1]^d$ be a point set such that $\text{vol}_{\text{opt}}(S) \geq \mu$, where μ is some constant, and let C_{opt} be the corresponding largest empty convex body in $[0, 1]^d$. Given a grid $\mathcal{G}(m)$, we call $\text{conv}(C_{\text{opt}} \cap \mathcal{G}(m))$ the *discrete hull* of C_{opt} [14]. We need the following easy lemma.

Lemma 3. *Let $C \subseteq [0,1]^d$ be a convex body and $D = \mathrm{conv}(C \cap \mathcal{G}(m))$. Then we have $\mathrm{vol}(C) - \mathrm{vol}(D) = O(1/m)$, where the constant of proportionality depends only on d.*

Proof. Consider a point $p \in C \setminus D$ and the set of $2d$ points $X = \{p \pm 2(d/m)e_i\}$, where e_i is the unit vector having one in the ith coordinate, and 0 everywhere else. If one of the points of X is outside C, then the distance from p to the boundary of C is at most $2d/m$. Otherwise, the cube $p + [-2,2]^d/m$ is contained in the "diamond" $\mathrm{conv}(X)$, which is in turn contained in C. But then, the grid points of the grid cell of $\mathcal{G}(m)$ containing p are in C, and p cannot be outside D. We reached a contradiction.

It follows that all the points of the corridor $C \setminus D$ are at distance at most $2(d/m)$ from the boundary of C. The volume of the boundary of C is bounded by the volume of the boundary of the unit cube, namely $2d$. As such, the volume of this corridor is $\mathrm{vol}(\partial C)\, O(d/m) \leq (2d)(2d/m) = O(d^2/m)$. For a fixed d, this is $O(1/m)$, as claimed. □

Lemma 3 implies that if $\mathrm{vol}_{\mathrm{opt}}(S) \geq \mu$, where μ is some constant, then we can concentrate our search on convex polytopes that have their vertices at grid points in $\mathcal{G}(m)$, where $m = O(1/\varepsilon\mu)$.

4.2 Finding a Large Empty Convex Polygon

We first re-derive a result of Eppstein et al. [9] concerning an exact algorithm for a related problem, with a simple proof.

Lemma 4. *Given a set S of n points and a set Q of m points in the plane, one can compute a convex polygon of the largest area with vertices in S that does not contain any point of Q in its interior in $O(n^3 m + n^4)$ time. The algorithm has the same running time if Q is a set of m forbidden rectangles.*

Proof. The algorithm works by dynamic programming. First, we compute for all triangles with vertices from S whether they contain a forbidden point inside them; trivially this can be done in $O(n^3 m)$ time. We then build a directed graph G on the allowable triangles, connecting two triangles Δ and Δ' if they share their left endpoint, are interior disjoint, share an edge, and their union forms a convex quadrilateral. We orient the edge from the triangle that is most counterclockwise (around the common vertex) to the other triangle. All edges are oriented "upwards", so G is a directed acyclic graph (DAG). Observe that G has $O(n^3)$ vertices (allowable triangles) and the maximum out-degree in G is bounded from above by n.

The weight of a vertex corresponding to a triangle is equal to its area. Clearly, a convex polygon corresponds to a path in G, namely the triangulation of the polygon from its leftmost vertex, and its weight is the area of the polygon. Finding the maximum weight path can be done in linear time in the size of the DAG; see e.g., [6, Section 4.7]. G has $O(n^3)$ vertices and $O(n^4)$ edges, and as such the overall running time is $O(n^3 m + n^4)$. □

Lemma 5. *Given a set $S \subseteq [0,1]^2$ of n points, such that $\mathrm{vol}_{\mathrm{opt}}(S) \geq \rho$, and a parameter $\varepsilon > 0$, one can compute an empty convex body $C \subseteq [0,1]^2$ such that $\mathrm{vol}(C) \geq (1-\varepsilon)\mathrm{vol}_{\mathrm{opt}}(S)$. The running time of the algorithm is $O\big(n + 1/(\varepsilon\rho)^8\big)$.*

Proof. Consider the grid $\mathcal{G}(m)$. By Lemma 3 we can restrict our search to a grid polygon. Going a step further, we mark all the grid cells containing points of S as forbidden. Arguing as in Lemma 3, one can show that the area of the largest convex grid polygon avoiding the forbidden cells is at least $\mathrm{vol}_{\mathrm{opt}}(S) - c/m$, where c is some constant.

We now restrict our attention to the task of finding this largest polygon. We have a set Q of $O(m^2)$ grid points that might be used as vertices of the grid polygon, and a set of $O(m^2)$ grid cells that can not intersect the interior of the computed polygon. Using Lemma 4 finding the largest empty polygon takes $O(m^8)$ time. Setting $m = 1/\varepsilon\rho$, we get an algorithm with overall running time $O\big(n + 1/(\varepsilon\rho)^8\big)$. □

4.3 The Higher Dimensional Case

Lemma 6. *Given a set $S \subseteq [0,1]^d$ of n points, such that $\mathrm{vol}_{\mathrm{opt}}(S) \geq \mu$, and a parameter $\varepsilon > 0$, one can compute an empty convex body $C \subseteq [0,1]^d$, such that $\mathrm{vol}(C) \geq (1-\varepsilon)\mathrm{vol}_{\mathrm{opt}}(S)$. The running time of the algorithm is $O\Big(n + m^{2d}2^{m^d}\Big)$, where $m = O(1/\varepsilon\mu)$.*

Proof. Consider the grid $\mathcal{G}(m)$. Let X be the set of all the grid cells of $\mathcal{G}(m)$ that contain points from S. Observe that $|X| = O(m^d)$. Next, let S' be the set of all grid points of $\mathcal{G}(m)$. Enumerating over all possible subsets of the grid points S' generates 2^{m^d} candidate sets. Checking if such a convex hull intersects the interior of a specific forbidden cell in X can be done in linear time, that is, in $O(m^d)$ time. Therefore, checking if such a candidate set convex hull is a valid solution, takes $O\big(m^{2d}\big)$ time. Returning the subset with the largest hull volume found yields the desired approximation. □

4.4 Better Approximations

Lemma 7. *Given a set $S \subseteq [0,1]^2$ of n points with $\mathrm{vol}_{\mathrm{opt}}(S) \geq \rho$ and parameters $\delta, \varepsilon > 0$, one can compute an empty convex body $C \subseteq [0,1]^2$ such that $\mathrm{vol}(C) \geq (1-\varepsilon)\mathrm{vol}_{\mathrm{opt}}(S)$ with probability at least $1-\delta$. The running time of the algorithm is $O\Big(\frac{n\log^2 n}{\varepsilon\rho}\big(\log n + \frac{1}{\varepsilon^8}\big)\log\frac{1}{\delta\rho}\Big)$. For a fixed δ, the running time is $O\big(n^2\log^3 n\,\varepsilon^{-1}\big(\log n + \varepsilon^{-8}\big)\big)$.*

Proof. Let $\mathcal{E}$ be an ellipse of maximum area contained inside $C_{\mathrm{opt}} = C_{\mathrm{opt}}(S)$. As suggested by Lemma 2, let $\mathcal{R}$ be a random sample of $O(1/\rho\log(1/\delta\rho))$ points from $[0,1]^2$. With probability $\geq 1-\delta$ this sample hits $\mathcal{E}/2$. The intuitive idea is now to guess a copy of $\mathcal{E}/2$ and center it at one of the points of $\mathcal{R} \cap \mathcal{E}/2$. In particular, let $p \in \mathcal{R}$ be the guess for the desired center of this ellipse. Naturally, to guess the ellipse itself, we need to guess the lengths of the two axes of $\mathcal{E}/2$, and their orientation.

Since the shortest axis of $\mathcal{E}/2$ has length at least $1/8n$, and the maximum length axis has length at most $\sqrt{2}$, it follows that if we want to guess the lengths of the two axes, up to a factor of two, we need to consider only $O(\log^2 n)$ possibilities. Indeed, we consider the canonical lengths $\ell_i = 2^i/8n$, for $i = 0, \dots, \lceil \log_2(8n) \rceil$.

Consider now the bounding box of the guessed ellipse $\mathcal{F}$ (we do not know its orientation yet). Scale it up by a factor of 4 so that its contains $C_{\rm opt}$. Let B be the resulting box fixed in the right orientation. We can apply Lemma 5 to B (as the unit square) to get the desired approximation. The polygon $C_{\rm opt}$ occupies a constant fraction of the area of B, and as such the resulting running time is $O(n + 1/\varepsilon^8)$. Note that the algorithm of Lemma 5 partitions B into a grid with $O(1/\varepsilon^2)$ cells. The approximation algorithm cares only about which cells are empty or not.

Since we do not know the orientation of B, perform a rotational sweeping algorithm [5], rotating B around p. Whenever a point of S moves form one grid cell to another in the grid of B, stop and recompute the optimal solution. We have $O(n/\varepsilon)$ such events during the sweeping process, and an update requires $O(1/\varepsilon^8)$ time to handle. Hence the running time for computing this polygon for p is $O((n/\varepsilon)\log n + n/\varepsilon^9)$.

Since we have to repeat this for all the points in the random sample $\mathcal{R}$, and all ellipse axes, the overall running time is

$$O\left(\frac{n \log^2 n}{\varepsilon\rho}\left(\log n + \frac{1}{\varepsilon^8}\right)\log\frac{1}{\delta\rho}\right).$$

Since $\rho = \Omega(1/n)$, for a fixed δ, the above expression is bounded by $O\big(n^2 \log^3 n\, \varepsilon^{-1}\big(\log n + \varepsilon^{-8}\big)\big)$, as claimed. □

By doing an exponentially decreasing search for ρ, the running time increases only by a constant factor (this is a geometrically decreasing series, hence the term with the last value of ρ dominates the whole running time). We summarize our result for the plane in the following. Observe that if $\rho = \Omega(1)$ the running time of this planar algorithm is near linear in n.

Theorem 4. *Given a set $S \subseteq [0,1]^2$ of n points and parameters $\varepsilon, \delta > 0$, one can compute an empty convex body $C \subseteq [0,1]^2$, such that $\mathrm{vol}(C) \geq (1-\varepsilon)\mathrm{vol}_{\rm opt}(S)$ with probability at least $1-\delta$. The running time of the algorithm is $O\Big(\frac{n\log^2 n}{\varepsilon\rho}\big(\log n + \frac{1}{\varepsilon^8}\big)\log\frac{1}{\delta\rho}\Big)$, where $\rho = \mathrm{vol}_{\rm opt}(S)$. For a fixed δ, the running time is $O\big(n^2\log^3 n\,\varepsilon^{-1}\big(\log n + \varepsilon^{-8}\big)\big)$.*

In higher dimensions we obtain the following (see [7] for proof details).

Theorem 3. *Given a set S of n points in $[0,1]^d$, one can $(1-\varepsilon)$-approximate the maximum volume empty convex body in $[0,1]^d$. The running time of the approximation algorithm is*

$$O\Big(n^{1+d(d-1)/2} 2^{O\left(1/\varepsilon^d\right)} \log^d n\Big).$$

References

1. Aichholzer, O., Krasser, H.: The point set order type data base: A collection of applications and results. In: Proc. 13th Canadian Conf. on Comput. Geom., Waterloo, pp. 17–20 (2001)
2. Alon, N., Bárány, I., Füredi, Z., Kleitman, D.: Point selections and weak ε-nets for convex hulls. Combinatorics, Probability & Computing 1, 189–200 (1992)
3. Bambah, R.P., Woods, A.C.: On a problem of Danzer. Pacific J. Math. 37(2), 295–301 (1971)
4. Beck, J., Chen, W.: Irregularities of Distributions. Cambridge Tracts in Mathematics, vol. 89. Cambridge University Press, Cambridge (1987)
5. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer (2008)
6. Dasgupta, S., Papadimitriou, C., Vazirani, U.: Algorithms. McGraw-Hill, New York (2008)
7. Dumitrescu, A., Har-Peled, S., Tóth, C.D.: Minimum convex partitions and maximum empty polytopes, arXiv:1112.1124
8. Dumitrescu, A., Tóth, C.D.: Minimum weight convex Steiner partitions. Algorithmica 60(3), 627–652 (2011)
9. Eppstein, D., Overmars, M., Rote, G., Woeginger, G.: Finding minimum area k-gons. Discrete Comput. Geom. 7(1), 45–58 (1992)
10. Fevens, T., Meijer, H., Rappaport, D.: Minimum convex partition of a constrained point set. Discrete Appl. Math. 109(1-2), 95–107 (2001)
11. Füredi, Z., Pach, J.: Traces of finite sets: extremal problems and geometric applications. In: Frankl, P., Füredi, Z., Katona, G., Miklós, D. (eds.) Extremal Problems for Finite Sets, Budapest. Bolyai Soc. Math. Studies, vol. 3, pp. 251–282 (1994)
12. García López, J., Nicolás, C.: Planar point sets with large minimum convex partitions. In: Proc. 22nd European Workshop on Comput. Geom., pp. 51–54 (2006)
13. Grantson, M., Levcopoulos, C.: A Fixed Parameter Algorithm for the Minimum Number Convex Partition Problem. In: Akiyama, J., Kano, M., Tan, X. (eds.) JCDCG 2004. LNCS, vol. 3742, pp. 83–94. Springer, Heidelberg (2005)
14. Har-Peled, S.: An output sensitive algorithm for discrete convex hulls. Comput. Geom. Theory Appl. 10, 125–138 (1998)
15. Hershberger, J., Suri, S.: Applications of a semi-dynamic convex hull algorithm. BIT 32(2), 249–267 (1992)
16. Horton, J.: Sets with no empty convex 7-gons. Canadian Math. Bulletin 26, 482–484 (1983)
17. Knauer, C., Spillner, A.: Approximation Algorithms for the Minimum Convex Partition Problem. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 232–241. Springer, Heidelberg (2006)
18. Matoušek, J.: Geometric Discrepancy: An Illustrated Guide. Springer (1999)
19. Matoušek, J.: Lectures on Discrete Geometry. Springer (2002)
20. Neumann-Lara, V., Rivera-Campo, E., Urrutia, J.: A note on convex partitions of a set of points in the plane. Graphs and Combinatorics 20(2), 223–231 (2004)
21. Sakai, T., Urrutia, J.: Convex partitions of point sets in the plane. In: Proc. 7th Japan Conf. on Comput. Geom. and Graphs, Kanazawa. JAIST (2009)
22. Spillner, A.: A fixed parameter algorithm for optimal convex partitions. J. Discrete Algorithms 6(4), 561–569 (2008)
23. Valtr, P.: Sets in $\mathbb{R}^d$ with no large empty convex subsets. Discrete Mathematics 108(1-3), 115–124 (1992)

A Probabilistic Analysis of Christofides' Algorithm*

Markus Bläser[1], Konstatinos Panagiotou[2], and B.V. Raghavendra Rao[1]

[1] Department of Computer Science, Saarland University
{mblaeser,bvrr}@cs.uni-saarland.de
[2] Department of Mathematics, University of Munich
kpanagio@math.lmu.de

Abstract. Christofides' algorithm is a well known approximation algorithm for the metric travelling salesman problem. As a first step towards obtaining an average case analysis of Christofides' algorithm, we provide a probabilistic analysis for the stochastic version of the algorithm for the Euclidean traveling salesman problem, where the input consists of n randomly chosen points in $[0,1]^d$. Our main result provides bounds for the length of the computed tour that hold almost surely. We also provide an experimental evaluation of Christofides's algorithm.

1 Introduction

The Traveling Salesman Problem, TSP for short, is a well-known NP-hard combinatorial optimization problem. For general edge weights, it is even NP-hard to find any sub-exponential approximation, see e.g. [10]. One natural restriction is the case where the edge weights fulfill the triangle inequality. The problem remains NP-hard (and APX-hard) for this restriction as well, but constant factor approximation algorithms are well known, like Christofides' algorithm [6] or the tree doubling algorithm [7]. Euclidean Traveling Salesman Problem (ETSP for short) is the restriction of metric TSP, where the vertices of the graph are points in $\mathbb{R}^d$ and the edge weights are the Euclidean distances between them. ETSP is also NP-hard to compute exactly, however efficient approximation schemes are known [1,13].

There has been a lot of interest to understand the asymptotic behavior of Euclidean combinatorial optimization problems, in particular ETSP. In their seminal paper [4], Beardwood, Halton and Hammersley performed a probabilistic analysis, where they showed the following remarkable result:

Theorem 1. *Let $d \geq 2$. Let $U_1, \ldots, U_n$ be n independent uniformly distributed random points over $[0,1]^d$. There exists a constant $\gamma_{\mathsf{ETSP}} = \gamma_{\mathsf{ETSP}}(d) > 0$ such that almost surely*

$$\lim_{n\to\infty} \frac{\mathsf{ETSP}(U_1, \ldots, U_n)}{n^{(d-1)/d}} = \gamma_{\mathsf{ETSP}}.$$

* This project is partially funded by the DFG grant BL 511/7-1. The work was performed while the second author was affiliated with Max Planck Institute for Informatics in Saarbrücken.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 225–236, 2012.

In words, the authors of [4] showed that when we draw n points uniformly at random from $[0, 1]^d$, the cost of an optimal tour is almost surely asymptotically equal to $\gamma_{\mathsf{ETSP}}\, n^{(d-1)/d}$. Later on, this result was generalized to many other problems on Euclidean spaces, like the minimum spanning tree problem [17]. In particular, Steele [16] provided a general framework that provides similar results for all Euclidean functionals that are sub-additive (see Definition 1).

Motivated by the result of Beardwood *et al.*, Karp [12] gave a partitioning heuristic for ETSP that runs in polynomial time and asymptotically outputs an optimal tour with probability 1 over points uniformly sampled from $[0, 1]^d$. Starting with Karp's work, there has been a lot of interest in the probabilistic analysis of heuristic algorithms for Euclidean optimization problems. For instance, Avis, Davis, and Steele [2] showed complete convergence for the greedy algorithm for the Euclidean minimum matching problem and Goemans and Bertsimas [9] provided the almost sure asymptotics for the Held-Karp relaxation of ETSP. For an overview of further results, we refer to [8,18,19,3].

In this paper we study Christofides' algorithm, a popular heuristic for metric TSP. It starts with computing a minimum spanning tree and then a minimum matching on the odd-degree vertices. In the resulting graph, every node has even degree and therefore, the graph is Eulerian. We obtain a TSP tour by taking shortcuts in the Eulerian tour. For arbitrary metrics, Christofides' algorithm achieves a 3/2-approximation.

Despite its worst case approximation ratio of 3/2, Christofides' algorithm is known to perform better in practice. Analyzing Christofides' algorithm on random point sets has been an open problem, as posed by Frieze and Yukich in 2002 [8]. In this work, we introduce the functional CHR as the sum of cost of a minimum spanning tree and a minimum matching on the odd-degree vertices of the minimum spanning tree (see Section 3). Clearly, CHR serves as a worst case upper bound on the cost of the tour computed by Christofides' algorithm. We prove:

Theorem 2. *Let $d \geq 2$. Let $U_1, \ldots, U_n$ be n i.i.d. uniformly distributed random points over $[0, 1]^d$. There exists a constant $\gamma_{\mathsf{CHR}} = \gamma_{\mathsf{CHR}}(d) > 0$ such that almost surely*

$$\lim_{n\to\infty} \frac{\mathsf{CHR}(U_1, \ldots, U_n)}{n^{(d-1)/d}} = \gamma_{\mathsf{CHR}}.$$

As a main ingredient in our proof, we show that the functional CHR satisfies a weak form of geometric sub-additivity (see Definition 2). Then we show that the techniques developed by Steele [16] can be extended to weakly sub-additive functionals.

Note that this result for Christofides' algorithm is not a consequence of the results known for Euclidean minimum spanning trees and Euclidean minimum matching, for the matching computed by the Christofides' algorithm depends on the odd degree vertices in the minimum spanning tree. Moreover, it is not clear if Christofides' functional is sub-additive, and thus it is not possible to apply known methods. However, we show that Christofides' functional fulfills a weaker property, which is still strong enough to obtain the desired results. A related approach to overcome the limitations of the classical methods was taken by Baltz *et al.* [3], who studied a routing problem with multiple depots.

Before we proceed with the proofs we comment on the values of the constants γ_{ETSP} and γ_{CHR} in Theorems 1 and 2. It is well-known that the worst-case approximation ratio of Christofides' algorithm is $3/2$. On the other hand, the theorems above imply that the approximation ratio is at most $\gamma_{\mathsf{CHR}}/\gamma_{\mathsf{TSP}}$ on almost all pointsets. It is an intriguing question whether this ratio is $< 3/2$. However, a numerical or analytical evaluation seems from a current perspective very difficult: although numerous efforts have been made in the past, see e.g. [14,5,11] and many references therein, the constant γ_{ETSP} is not known exactly. Similarly, as in all previous proofs regarding the asymptotic behavior of Euclidean functionals, our proof does not provide any way of computing the value of γ_{CHR}. On the positive side, our experimental evaluation on random points shows that γ_{CHR} is strictly below 1.5, even without shortcutting (see Section 5).

2 Preliminaries

Most of the notations used here are from [19]. In this paper the distance between two points in $[0,1]^d$ is always the Euclidean distance.

Euclidean Functionals. Let $d > 1$ be a fixed dimension. An *Euclidean functional* in dimension d is a function f that maps any finite point set $X \subset \mathbb{R}^d$ to a positive real number $f(X)$. We use the following Euclidean functionals in the paper:

- $\mathsf{TSP}(X)$ = total edge weight of a minimum Euclidean traveling salesman tour of X.
- $\mathsf{MM}(X)$ = total edge weight of a minimum weight Euclidean perfect matching of X. (If $|X|$ is odd, then one point will be left unmatched.)
- $\mathsf{MST}(X)$ = total edge weight of a minimum Euclidean spanning tree of X.

Following standard notation, for an Euclidean functional f and a hypercube $\mathcal{H} \subset \mathbb{R}^d$, $f(.,\mathcal{H})$ will denote the restriction of f, when the points are restricted to $\mathcal{H}$. In other words, for any point set X we have that $f(X,\mathcal{H}) = f(X \cap \mathcal{H})$. We define certain properties of Euclidean functionals that will be used throughout.

An Euclidean functional f is *monotone* if for every $F \subseteq G$, $f(F,\mathcal{H}) \leq f(G,\mathcal{H})$. Moreover, f is said to be *homogeneous* if

$$\forall\, \alpha > 0: \;\; f(\alpha\, F, \alpha\, \mathcal{H}) = \alpha \cdot f(F,\mathcal{H}),$$

where $\alpha\, X = \{\alpha\, x : x \in X\}$ for any $X \subseteq \mathbb{R}^d$. We will say that f is *translation invariant* if

$$\forall\, a \in \mathbb{R}^d: \;\; f(F,\mathcal{H}) = f(F + a, \mathcal{H} + a)$$

where $X + a = \{x + a : x \in X\}$ for $X \subseteq \mathbb{R}^d$. We say that f admits a *growth bound*, if there is a constant $C > 0$ such that

$$f(F,[0,1]^d) \leq C|F|^{(d-1)/d}, \tag{1}$$

Finally, f is called *smooth* if there is a constant C such that for all $F, G \subset [0,1]^d$,

$$|f(F \cup G, [0,1]^d) - f(F, [0,1]^d)| \leq C|G|^{(d-1)/d}. \tag{2}$$

A further important property of Euclidean functionals is *sub-additivity.*

Definition 1 (Sub-additivity). *Let $Q_1, \ldots, Q_{m^d}$ be a partition of $[0,1]^d$ into equal-sized sub-cubes of edge m^{-1} each. Then f is sub-additive if there is a $C = C(d) \geq 0$ such that for $m \in \mathbb{Z}^+$,*

$$f(F, [0,1]^d) \leq \sum_{i=1}^{m^d} f(F, Q_i) + Cm^{d-1}. \tag{3}$$

Sub-additivity is the one of the most important properties used in several studies of Euclidean functionals, in particular $\mathsf{TSP}, \mathsf{MM}$, and MST, see [8,19] for an excellent survey.

Proposition 1. *The Euclidean functionals* $\mathsf{TSP}, \mathsf{MM}$*, and* MST *are homogeneous, translation invariant, smooth, sub-additive, and admit a growth bound.*

The Christofides' functional is not sub-additive. However, we define a weaker property that turns out to be sufficient for our analysis of the Christofides' functional.

Definition 2 (Weak Sub-additivity). *Let $Q_1, \ldots, Q_{m^d}$ be a partition of the unit box $[0,1]^d$ into equal-sized sub-cubes of edge m^{-1} each. Then, f is said to be weakly sub-additive, if there are constants $C = C(d), C' = C'(d) \geq 0$ and $\epsilon = \epsilon(d) > 0$ such that for $m \in \mathbb{Z}^+$,*

$$f(F, [0,1]^d) \leq \sum_{i=1}^{m^d} f(F, Q_i) + Cn^{((d-1)/d)-\epsilon}m^{\epsilon} + C'm^{d-1}. \tag{4}$$

We will use the following (folklore) facts about Euclidean minimum matching. We give a simple proof for completeness.

Lemma 1. *Let $A, B \subset [0,1]^d$ be two finite, disjoint sets of points with even cardinality. Then*

$$\mathsf{MM}(B) \leq \mathsf{MM}(A \cup B) + \mathsf{MM}(A)$$

Proof. Let M be a minimum matching of $A \cup B$ and M_1 be that of A. Consider the graph $G = (A \cup B, M_1 \cup M_2)$. For every $u \in B$, there is a unique path P_u originating at u. Moreover, P_u ends at some $u' \in B$, and all the remaining vertices in P are from A. This gives a matching for B of the required cost.

Lemma 2 (Folklore). *Let T be a MST of n points. Then the cost of minimum matching on the odd-degree vertices of T is bounded by* $\mathsf{MST}(T)$.

Further Notation. If S is any collection of edges in a graph, and v is a vertex then $\varDelta_S(v)$ denotes the degree of v in the sub-graph induced by S. Moreover, we will write $\|S\|$ for the sum of the lengths of the edges in S.

3 Proof of the Main Result

In this section we present the main steps that are needed to achieve the proof of Theorem 2. We begin with defining the Euclidean functional given by Christofides algorithm. For any point set $F \subset [0,1]^d$ let $\mathsf{CHR}(F)$ denote the cost of a minimum spanning tree T of the points in F plus the cost of a minimum matching of the odd-degree points in T. In symbols, we write

$$\mathsf{CHR}(F) \triangleq \mathsf{MST}(F) + \mathsf{OM}(F)$$

where $\mathsf{OM}(F)$ denotes the cost of a minimum matching on the odd degree vertices in the minimum spanning tree. When F has more than one minimum spanning trees, $\mathsf{CHR}(F)$ is defined as the minimum over all such trees. However, we will not consider such exceptional cases in our analysis, since the spanning tree of a random point set is unique.

Note that strictly speaking, the functional CHR defined above does not measure the length of the tour obtained by Christofides' algorithm, as we ignore shortcuts. This is done in order to have more structure in the functional CHR, even though it weakens the analysis a bit.

The first step in our proof is to establish the following lemma about the structure of the functional CHR. It is the main contribution of our paper, and it is proved in Section 4.

Lemma 3. *The Euclidean functional* CHR *is homogeneous, translation invariant, smooth, and admits a growth bound. Moreover, it is weakly sub-additive.*

With this fact at hand, we proceed with proving a general result that determines the asymptotic value of the expectation of a weakly sub-additive Euclidean functional. This result, together with the proof, are generalizations of the corresponding theorems that hold for sub-additive functionals only, and thus they can be applied to a wider class of functions. Due to space limitations, the proof is omitted.

Theorem 3. *Let $d \geq 2$. Let f be a smooth, weakly sub-additive Euclidean functional that admits a growth bound. There is a $\gamma_f = \gamma_f(d)$ such that if $U_1, \ldots, U_n$ are uniform i.i.d over $[0,1]^d$, then*

$$\lim_{n\to\infty} \frac{\mathbb{E}\left[f(U_1,\ldots,U_n)\right]}{n^{(d-1)/d}} = \gamma_f .$$

Together with Lemma 3, the above theorem implies that there is a $\gamma_{\mathsf{CHR}} \geq 0$ such that

$$\mathbb{E}\left[\mathsf{CHR}(U_1,\ldots,U_n)\right] = (1+o(1)) \cdot \gamma_{\mathsf{CHR}} \cdot n^{(d-1)/d}. \tag{5}$$

However, as $\gamma_{\mathsf{MST}} > 0$, see [8], we also obtain that $\gamma_{\mathsf{CHR}} > 0$.

Note that the above collection of arguments almost shows Theorem 2. To complete the proof it remains to show that $\mathsf{CHR}(U_1,\ldots,U_n)$ is typically very close to its expected value. This is performed by the next well-known result, which follows immediately from the arguments exposed in [15].

Theorem 4. *Let f be a homogeneous, translation invariant and smooth Euclidean functional. Suppose that there is a $\gamma_f = \gamma_f(d)$ such that*

$$\lim_{n\to\infty} \frac{\mathbb{E}[f(U_1,\dots,U_n)]}{n^{(d-1)/d}} = \gamma_f.$$

Then there is a $C = C(d) > 0$ such that for sufficiently large n

$$\mathbb{P}\left[|f(U_1,\dots,U_n) - \mathbb{E}[f(U_1,\dots,U_n)]| > t\right] \le \exp\left\{-C\frac{t^{2d/(d-1)}}{n}\right\}.$$

Theorem 2 is then a direct consequence of (5) together with Lemma 3 and the above result, which we apply with, say, $t = n^{2(d-1)/3d} = o(n^{(d-1)/d})$.

4 Christofides' Functional

This section is devoted to the proof of Lemma 3. We start with proving all the properties except for the weak sub-additivity of it.

Lemma 4. CHR *is a homogeneous, translation invariant, smooth Euclidean functional that admits a growth bound.*

Proof. As translation or scaling does not change the relative distances between the points, CHR is homogeneous and translation invariant. The growth bound can be obtained by that of MST and minimum matching, see [8,19].

We now argue that CHR is also smooth. Let $D(d)$ denote the bound on the maximum degree in any Euclidean minimum spanning tree of a d-dimensional pointset. Note that $D(d)$ depends only on d. Let $F, G \subset [0,1]^d$ be any finite sets of points. Let T be a minimum spanning tree of F and $O \subseteq F$ denote the set of odd-degree points in T. Let T' be the minimum spanning tree of $F \cup G$, obtained by iteratively adding points from G, one at a time, and then adding/removing necessary edges to/from T. Let us examine the first step in the above procedure. Let $v \in G$, and let T_1 be a minimum spanning tree of $F \cup \{v\}$. Then by the degree bound on the Euclidean minimum spanning tree, v can have at most $D(d)$ incident edges in any MST of $F \cup \{v\}$. For each such edge there is at most one edge in T that has to be removed to ensure the acyclicity of T_1. So, each edge incident to v can affect the degrees of at most 3 points. Thus, the degrees of at most $3D(d)$ points in T_1 are different from that in T, and we infer that in total at most $3D(d)|G|$ points will have their degrees in T' different from that in T. Let O' denote the set of odd degree vertices in T'. The above discussion guarantees that

$$||O| - |O'|| \le 3D(d)|G|.$$

As $O \cap O'$ and $O \setminus O'$ form a partition of O, we have:

$$\mathsf{MM}(O) \le \mathsf{MM}(O \cap O') + \mathsf{MM}(O \setminus O') + t_1, \tag{6}$$

where t_1 is the cost of a single edge if $|O \cap O'|$ is odd and zero otherwise. By Lemma 1 with $B = O \cap O'$, $A = O' \setminus (O \cap O')$ (hence $A \cup B = O'$),

$$\mathsf{MM}(O \cap O') \leq \mathsf{MM}(O') + \mathsf{MM}\left(O' \setminus (O \cap O')\right).$$

Substituting in (6), and using that $|O \setminus O'|, |O' \setminus (O \cap O')| \leq 3D(d)|G|$ and $\mathsf{MM}(X) = O(|X|^{(d-1)/d})$ for any pointset $X \subset [0,1]^d$

$$\begin{aligned}\mathsf{MM}(O) &\leq \mathsf{MM}(O') + \mathsf{MM}(O \setminus O') + \mathsf{MM}\left(O' \setminus (O \cap O')\right) + t_1 \\ &\leq \mathsf{MM}(O') + 3(3D(d)|G|)^{(d-1)/d}\end{aligned}$$

By interchanging the roles of O and O' in the above argument, we can show similarly that

$$\mathsf{MM}(O') \leq \mathsf{MM}(O) + \mathsf{MM}(O' \setminus O) + \mathsf{MM}(O \setminus O') + t_2$$

where t_2 is the cost of a single edge in $[0,1]^d$. Hence,

$$\mathsf{MM}(O') \leq \mathsf{MM}(O) + 3(3D(d)|G|)^{(d-1)/d}$$

Thus we have shown that there is a $C = C(d) > 0$ such that

$$\mathsf{OM}(F) - C|G|^{(d-1)/d} \leq \mathsf{OM}(F \cup G) \leq \mathsf{OM}(F) + C|G|^{(d-1)/d}. \tag{7}$$

By the definition of CHR and the triangle inequality, we have

$$|\mathsf{CHR}(F) - \mathsf{CHR}(F \cup G)| \leq |\mathsf{MST}(F) - \mathsf{MST}(F \cup G)| + |\mathsf{OM}(F) - \mathsf{OM}(F \cup G)|.$$

As MST is a smooth functional, $|\mathsf{MST}(F) - \mathsf{MST}(F \cup G)| = O(|G|^{(d-1)/d})$. Combining this with (7) then proves the claim.

We cannot show that CHR is sub-additive. However, we show that it satisfies a weaker form of subadditivity, which, however, is sufficient for our purposes.

Lemma 5. CHR *is weakly sub-additive.*

Before proving Lemma 5, we prove some of the structural properties of Euclidean minimum spanning trees and minimum matchings.

Notation. We use the following notation in Lemmas 6 and 7. Let T be a minimum spanning tree of a finite point set $F \subset [0,1]^d$. Let $Q_1, \ldots Q_{m^d}$ be the partitioning of $[0,1]^d$ into sub-cubes side length m^{-1} each. An edge $e = (u,v)$ in T is called a *boundary edge*, if $u \in Q_i$ and $v \in Q_j$, where $i \neq j$. A boundary edge (u,v) of T is called *short*, if $Q_i \cap Q_j \neq \emptyset$. We shall say that Q_i and Q_j are adjacent in this case. Every boundary edge that is not short will be denoted as *long*. A point $v \in F$ is said to be a *boundary point*, if it is incident to at least one of the boundary edges of T. Let B denote the set of boundary points of T that are incident on short edges. Let $B_i = B \cap Q_i$. Let $r \leq m^{-1}$ be a parameter to be chosen later. Let Q_i and Q_j be two adjacent sub-cubes, and $\mathcal{B}_{i,j}$ denote the boundary between them (i.e, a sub-cube in dimension at most $d-1$). Let $C_1, \ldots, C_t$ denote the partitioning of $\mathcal{B}_{i,j}$ into sub-cubes of side length r each. As $\sum_{k=1}^{t} \mathsf{Vol}(C_i) = \mathsf{Vol}(\mathcal{B}_{i,j}) \leq m^{-(d-1)}$, we have

$$t \leq m^{-(d-1)}/r^{d-1}. \tag{8}$$

For $1 \leq k \leq t$, let $\mathcal{C}_{i,k}$ (resp. $\mathcal{C}_{j,k}$) denote the hyper-rectangle in Q_i (resp. Q_j) with C_k as one of its base face.

Lemmas 6 and 7 provide some structural properties of B. The proof of the next statement is not very difficult and can be found in the Appendix.

Lemma 6. *With the notations above, suppose that $\overline{AB}$ and $\overline{CD}$ are two boundary edges such that $A, D \in \mathcal{C}_{i,k}$ and $B, C \in \mathcal{C}_{j,k'}$ for some $1 \leq k, k' \leq t$. Then, at least one point each from $\{A, D\}$ and $\{B, C\}$ is at distance at most $\sqrt{d}r$ to $\mathcal{B}_{i,j}$.*

Corollary 1. *Let Q_i and Q_j be two adjacent sub-cubes. Then there are at most t^2 boundary edges between points in Q_i and Q_j of length more than $2\sqrt{d}r$.*

Proof. By Lemma 6, for a sub-rectangle $\mathcal{C}_{i,k}$, there are at most t boundary points in Q_j at a distance of at least $\sqrt{d}r$ from the boundary. As there are t such rectangles $\mathcal{C}_{i,k}$, we get the desired bound.

In Lemma 7 below, we bound the cost of a minimum spanning tree or a minimum matching for the points in B_i. (See Appendix for a proof.)

Lemma 7. *Suppose $n_i = |Q_i \cap F| \geq 1$. There is an $\epsilon = \epsilon(d) > 0$ such that the cost of a minimum matching or a minimum spanning tree of any subset of points in B_i is $O(m^{-1} n_i^{((d-1)/d)-\epsilon})$.*

We also bound the total edge length of long boundary edges in T. (See Appendix for a proof.)

Lemma 8. *The total length of all long boundary edges in T is $O(m^{d-1})$.*

Proof (of Lemma 5). Let T be a minimum spanning tree of F. Let O denote the set of odd degree points in T. Let T_i denote the restriction of T to Q_i obtained by removing the edges incident to points outside Q_i. Let T_i' denote a minimum spanning tree for $F_i = F \cap Q_i$ obtained by adding necessary edges to T_i. Let $S_i = E(T_i') \setminus E(T_i)$. Let O_i' denote the set of odd degree points in T_i'. Let $O_i = O \cap Q_i$. By definition, $\mathsf{CHR}(F, [0,1]^d) = \|T\| + \mathsf{MM}(O)$. We need to prove

$$\mathsf{CHR}(F, [0,1]^d) \leq \sum_{i=1}^{m^d} \mathsf{CHR}(F_i, Q_i) + O(n^{((d-1)/d)-\epsilon} m^{\epsilon} + m^{d-1}). \tag{9}$$

for some $\epsilon = \epsilon(d) > 0$. Applying the geometric sub-additivity of the Euclidean minimum spanning tree functional, and that of Euclidean minimum matching,

$$\mathsf{CHR}(F, [0,1]^d) \leq \sum_{i=1}^{m^d} \|T_i'\| + \sum_{i=1}^{m^d} \mathsf{MM}(O_i) + O(m^{d-1}). \tag{10}$$

By the definition of CHR, we have $\mathsf{CHR}(F_i) = \|T_i'\| + \mathsf{MM}(O_i')$. Thus it is sufficient to bound $\mathsf{MM}(O_i)$ in terms of $\mathsf{MM}(O_i')$. This is performed by the next claim.

Claim 5 $\mathsf{MM}(O_i) \leq \mathsf{MM}(O'_i) + \|S_i\| + O(m^{-1}n_i^{((d-1)/d)-\epsilon} + m^{-1})$.

Applying the above claim on (10),

$$\mathsf{CHR}(F, [0,1]^d) \leq \sum_{i=1}^{m^d} \Big(\|T'_i\| + \mathsf{MM}(O'_i) + \|S_i\| + C' m^{-1} n_i^{((d-1)/d)-\epsilon} + O(m^{-1}) \Big)$$

The crucial observation is that it is sufficient to replace $\|T'_i\|$ by $\|T_i\|$ with a small additive term in the above sum, since $\|T'_i\| = \|T_i\| + \|S_i\|$. This can be done using Lemma 7. We prove

$$\|T'_i\| \leq \|T_i\| + O(m^{-1}n_i^{((d-1)/d)-\epsilon} + m^{-1})$$

To see this, note first that if T_i is connected, then $\|S_i\| = 0$, hence assume that T_i is not connected. Then at least one point in each of the connected components of T_i is a boundary point. So, T_i plus a spanning tree of all boundary points in F_i and a single edge connecting them gives a spanning tree τ_i of F_i, and hence $\|T'_i\| \leq \|\tau_i\|$. The boundary points in F_i can be partitioned into B_i, and the remaining boundary points of F_i that are incident on long boundary points. By Lemma 7, we have $\mathsf{MST}(B_i) \leq O(m^{-1}n_i^{((d-1)/d)-\epsilon}) + O(m^{-1})$. The boundary points that are incident on long boundary points can be connected arbitrarily to each other, as their total length is at most $O(m^{d-1})$ by Lemma 8. Thus,

$$\|T'_i\| \leq \|\tau_i\| \leq \|T_i\| + \mathsf{MST}(B_i) + O(m^{-1}) + \mathsf{MST}(\text{long boundary points in } F_i)$$

Thus there is a constant $C_1 = C_1(d) \geq 0$ such that,

$$\begin{aligned}\|T\| &\leq \sum_{i=1}^{m^d} \|T_i\| + C_1 \cdot (n_i^{((d-1)/d)-\epsilon} + m^{-1}) + \mathsf{MST}(\text{long boundary points in } F_i) \\ &\leq \sum_{i=1}^{m^d} \big(\|T_i\| + C_1 \cdot (n_i^{((d-1)/d)-\epsilon} + m^{-1})\big) + O(m^{d-1}).\end{aligned}$$

Hence, there is a $C'' = C''(d) \geq 0$ such that

$$\begin{aligned}\mathsf{CHR}(F, [0,1]^d) &= \|T\| + \mathsf{MM}(O) \\ &\leq \sum_{i=1}^{m^d} \Big(\|T_i\| + C_1 \cdot (m^{-1}n_i^{((d-1)/d)-\epsilon} + m^{-1}) \Big) \\ &\quad + \sum_{i=1}^{m^d} \Big(\mathsf{MM}(O'_i) + \|S_i\| + C' m^{-1} n_i^{((d-1)/d)-\epsilon} + O(m^{-1}) \Big) \\ &= \sum_{i=1}^{m^d} \Big(\|T_i\| + \|S_i\| + \mathsf{MM}(O'_i) + C'' m^{-1} n_i^{((d-1)/d)-\epsilon} + O(m^{-1}) \Big)\end{aligned}$$

As $\|T_i'\| = \|T_i\| + \|S_i\|$ and $\mathsf{CHR}(F, Q_i) = \|T_i'\| + \mathsf{MM}(O_i')$, we have that

$$\begin{aligned}\mathsf{CHR}(F, [0,1]^d) &\leq \sum_{i=1}^{m^d} \Big\{\mathsf{CHR}(F_i, Q_i) + C''(m^{-1} n_i^{((d-1)/d)-\epsilon} + m^{-1})\Big\} \\ &\leq \sum_{i=1}^{m^d} \mathsf{CHR}(F_i, Q_i) + C_2 \big(\sum_i n_i\big)^{((d-1)/d)-\epsilon} m^{\epsilon} \\ &\quad \text{(Hölder's inequality)} \\ &= \sum_{i=1}^{m^d} \mathsf{CHR}(F_i, Q_i) + C_2 n^{((d-1)/d)-\epsilon} m^{\epsilon}.\end{aligned}$$

To complete the proof of Lemma 5, we need to prove Claim 5. Due to space limitations, the proof is omitted and can be found in the full version of the paper.

5 Experimental Evaluation

In this section we present simulation results that shed some light on the actual values of the constants γ_{ETSP} and γ_{CHR}. In particular, we provide experimental evidence that the value of Christofides' functional is *strictly* less than 3/2 times the length of an optimal TSP tour through n random points.

Our experimental setup is as follows. Let $n_i = 500\,i$, where $1 \leq i \leq 20$. For any i in the given range, we generated independently 100 sets of n_i uniformly distributed random points in $[0,1]^2$, and computed the average and the standard

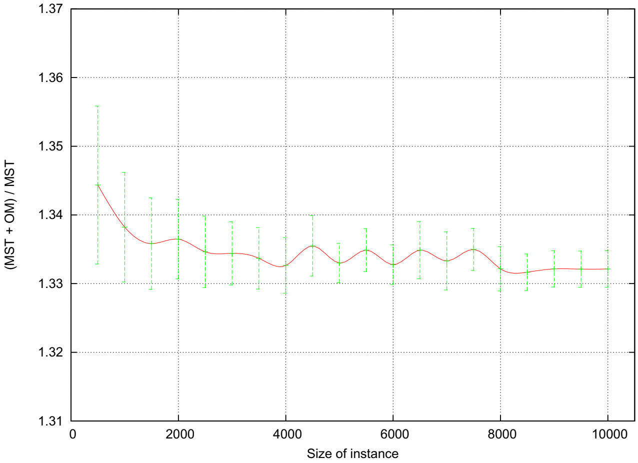


Fig. 1. An experimental upper bound for the value of the Christofides' functional divided by the length of a optimum TSP tour

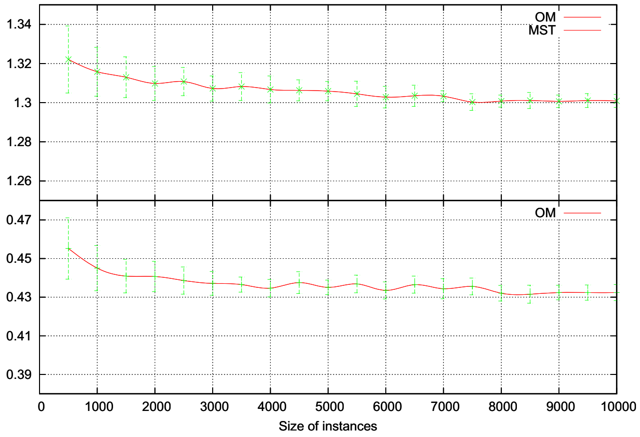


Fig. 2. The length of MST and OM

deviation of three parameters: i) the size of a minimum spanning tree (MST), ii) the size of a minimum matching on the odd degree vertices (OM) of the minimum spanning tree, and iii) the ratio (MST + OM) / MST. Note that the latter is an upper bound for the approximation ratio of Christofides' algorithm, since the length of a minimum spanning tree is a lower bound for the length of a TSP tour.

The results of the experiments are summarized in Figures 1 and 2, and lead to the following conclusions. Observe that the ratio (MST + OM) / MST stabilizes quickly around approximately 1.3347, and the standard deviation becomes small very quickly. In other words, even if we do not perform any shortcutting, the approximation ratio stays well below the worst-case value 3/2.

References

1. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J. ACM 45(5), 753–782 (1998)
2. Avis, D., Davis, B., Steele, J.M.: Probabilistic analysis of a greedy heuristic for euclidean matching. Probability in the Engineering and Informational Sciences 2(02), 143–156 (1988)
3. Baltz, A., Dubhashi, D.P., Srivastav, A., Tansini, L., Werth, S.: Probabilistic analysis for a multiple depot vehicle routing problem. Random Struct. Algorithms 30(1-2), 206–225 (2007)
4. Beardwood, J., Halton, J.H., Hammersley, J.M.: The shortest path through many points. Proc. Cambridge Philos. Soc. 55, 299–327 (1959)
5. Bertsimas, D.J., van Ryzin, G.: An asymptotic determination of the minimum spanning tree and minimum matching constants in geometrical probability. Operations Research Letters 9(4), 223–231 (1990)
6. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie Mellon University (1976)

7. Deineko, V., Tiskin, A.: Fast minimum-weight double-tree shortcutting for metric tsp: Is the best one good enough? J. Exp. Algorithmics 14, 4.6–4.16 (2010)
8. Frieze, A.M., Yukich, J.E.: Probabilistic analysis of the traveling salesman problem. In: Gutin, G., Punnen, A. (eds.) The Traveling Salesman Problem and Its Variations, pp. 257–308. Kluwer Academic Publisher (2002)
9. Goemans, M.X., Bertsimas, D.J.: Probabilistic analysis of the held and karp lower bound for the euclidean traveling salesman problem. Mathematics of Operations Research 16(1), 72–89 (1991)
10. Gutin, G., Punnen, A.P. (eds.): The traveling salesman problem and its variations. Combinatorial Optimization, vol. 12. Kluwer Academic Publishers (2002)
11. Johnson, D.S., McGeoch, L.A., Rothberg, E.E.: Asymptotic experimental analysis for the held-karp traveling salesman bound. In: SODA 1996, pp. 341–350 (1996)
12. Karp, R.M.: Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. Math. of Operat. Research 2(3), 209–224 (1977)
13. Mitchell, J.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. SIAM J. Comput. 28(4), 1298–1309 (1999)
14. Rhee, W.T.: On the travelling salesperson problem in many dimensions. Random Struct. Algorithms 3(3), 227–234 (1992)
15. Rhee, W.T.: A matching problem and subadditive euclidean functionals. Ann. Appl. Probab. 3(3), 794–801 (1993)
16. Steele, J.M.: Subadditive Euclidean functionals and nonlinear growth in geometric probability. Ann. Probab. 9(3), 365–376 (1981)
17. Steele, J.M.: Growth rates of Euclidean minimal spanning trees with power weighted edges. Ann. Probab. 16, 1767–1787 (1988)
18. Steele, J.M.: Probability Theory and Combinatorial Optimization. CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 69. SIAM (1997)
19. Yukich, J.E.: Probability Theory of Classical Euclidean Optimization Problems. Lecture Notes in Mathematics, vol. 1675. Springer (1998)

New Approximation Algorithms for the Unsplittable Capacitated Facility Location Problem*

Babak Behsaz[1], Mohammad R. Salavatipour[1], and Zoya Svitkina[2]

[1] Dept. of Computing Sci., Univ. of Alberta, Edmonton, Alberta T6G 2E8, Canada
{behsaz,mrs}@ualberta.ca

[2] Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
zoya@cs.cornell.edu

Abstract. In this paper, we consider the Unsplittable (hard) Capacitated Facility Location Problem (UCFLP) with uniform capacities and present some new approximation algorithms for it. This problem is a generalization of the classical facility location problem where each facility can serve at most u units of demand and each client must be served by *exactly one* facility. It is known that it is NP-hard to approximate this problem within any factor without violating the capacities. So we consider bicriteria (α, β)-approximations where the algorithm returns a solution whose cost is within factor α of the optimum and violates the capacity constraints within factor β. We present a framework for designing bicriteria approximation algorithms and show two new approximation algorithms with factors $(10.173, 3/2)$ and $(30.432, 4/3)$. These are the first algorithms with constant approximation in which the violation of capacities is below 2. The heart of our algorithms is a reduction from the UCFLP to a restricted version of the problem. One feature of this reduction is that any $(O(1), 1+\epsilon)$-approximation for the restricted version implies an $(O(1), 1+\epsilon)$-approximation for the UCFLP for any constant $\epsilon > 0$ and we believe our techniques might be useful towards finding such approximations or perhaps $(f(\epsilon), 1+\epsilon)$-approximation for the UCFLP for some function f. In addition, we present a quasi-polynomial time $(1+\epsilon, 1+\epsilon)$-approximation for the (uniform) UCFLP in Euclidean metrics, for any constant $\epsilon > 0$.

Keywords: approximation algorithms, unsplittable capacitated facility location problem, Euclidean metrics.

1 Introduction

We consider the Unsplittable Capacitated Facility Location Problem (UCFLP) with uniform capacities. In this problem, we are given a set of clients C and facilities F where client j has demand d_j and each facility i has capacity u and

* Research supported by Alberta Innovates Future Technologies, Canada. The second author was additionally supported by NSERC.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 237–248, 2012.

opening cost f_i. We have a metric cost function c_{ij} which denotes the cost of serving one unit of demand of client j at facility i. The goal is to open a subset of facilities $I \subseteq F$ and assign each client j to *exactly one* open facility $\phi(j)$ to serve its entire demand d_j so that the total amount of demand assigned to each open facility is no more than u, while minimizing the total cost of opening facilities and connecting (serving) clients to them, *i.e.*, minimizing $\sum_{i \in I} f_i + \sum_{j \in C} d_j \cdot c_{\phi(j)j}$. This problem generalizes the bin packing, the minimum makespan, and some facility location problems. If the demands of clients can be served by multiple open facilities, then we have the *splittable* version of the problem (called splittable CFLP). If each facility can be opened multiple times then we have the so-called *soft-capacitated* version. Each of these relaxations (*i.e.*, allowing splitting the demands of clients and/or having multiple copies of each facility) makes the problem significantly easier as discussed below.

By a simple reduction from the partition problem, one can show that any approximation algorithm for the uniform UCFLP violates the capacities of $\Omega(n)$ facilities unless P=NP. Thus, research has focused on the design of bicriteria approximation algorithms. An (α, β)-*approximation* for the UCFLP returns a solution whose cost is within factor α of the optimum and violates the capacity constraints within factor β. It should be noted that if we violate capacity of a facility within factor β, we must pay β times its opening cost. In the context of approximation algorithms, Shmoys, Tardos, and Aardal [9] were the first to consider this problem and presented a $(9, 4)$-approximation algorithm. They used a filtering and rounding technique to get an approximation algorithm for the splittable version and used a rounding for the generalized assignment problem (GAP) [8] to obtain their algorithm for the unsplittable version. This technique of reducing the unsplittable version using the rounding for the GAP to the splittable version was a cornerstone of the subsequent approximation algorithms. Korupolu, Plaxton, and Rajaraman [5] gave the first constant factor approximation algorithm for the splittable hard capacitated version, and applied the GAP rounding technique of [9] to get a $(O(1), 2)$-approximation algorithm for the UCFLP. Applying the current best approximation algorithms for the splittable capacitated version with non-uniform capacities (*i.e.*, each facility has capacity u_i) [10] and uniform capacities [1], one can get factor $(11, 2)$ and $(5, 2)$ approximation algorithms for the UCFLP with non-uniform and uniform capacities, respectively.

Recently, Bateni and Hajiaghayi [3] modeled an assignment problem in content distribution networks by the UCFLP. In this application, it is crucial to keep the violation of capacities as small as possible. Motivated by this strict requirement on capacities, the authors of [3] designed a $(1 + \epsilon, 1 + \epsilon)$-approximation algorithm for *tree metrics* (for any constant $\epsilon > 0$) using a dynamic programming approach. They also presented a quasi-polynomial time $(1+\epsilon, 1+\epsilon)$-approximation algorithm (again for trees) for the non-uniform capacity case. Using Fakcharoenphol *et al.*'s improvement of Bartal's machinery this implies a polynomial time $(O(\log n), 1 + \epsilon)$-approximation algorithm for almost uniform capacities and a quasi-polynomial time $(O(\log n), 1+\epsilon)$-approximation algorithm for non-uniform case for an arbitrary constant $\epsilon > 0$.

All the known constant-factor algorithms for the UCFLP violate the capacity constraints by a factor of at least 2 which is mainly due to using the rounding algorithm for GAP [8]. Also, the algorithm of [3] (although has $1+\epsilon$ violation) is not a constant factor approximation. We present the first constant factor approximation algorithms with capacity violation factor less than 2. Particularly, we present two approximation algorithms with factors $(10.173, 3/2)$ and $(30.432, 4/3)$ for the UCFLP. We also consider the UCFLP restricted to Euclidean metrics and give a $(1+\epsilon, 1+\epsilon)$-approximation that runs in the quasi-polynomial time.

1.1 Related Works

Perhaps the most well-studied facility location problem is the *uncapacitated* facility location problem (UFLP). In this problem, we do not have the capacity constraints and we only need to decide which facilities to open; as each client will be assigned to its closest open facility. The first constant approximation for the UFLP was a 3.16-approximation algorithm by Shmoys, Tardos, and Aardal [9]. The ratio for the UFLP was improved in a series of papers down to 1.488 [6]. On the negative side, a result of Guha and Khuller [4], combined with an observation of Sviridenko implies 1.463-hardness for the UFLP.

Capacitated facility location problems have also received a lot of attention. The solutions of the soft capacitated version have a similar structure to the solution of uncapacitated version and this problem can be reduced to the UFLP. For example, see [7] for a reduction. This paper gives the current best ratio, 2, for the soft capacitated version to the best of our knowledge. Since Mahdian *et al.* [7] reduce the problem to the UFLP, they give a solution that sends each client to exactly one facility. As a result, this solution is also feasible for the unsplittable case and is a 2-approximation for this case too. This comes from the fact that the optimal value of splittable version is a lower-bound for the optimal value of the unsplittable version. In contrast, there is an important distinction between the splittable and unsplittable case in the presence of hard capacities, because even checking the feasibility of the latter becomes NP-hard and we need bicriteria algorithms for the latter (see discussions above). In a series of local search algorithms, the ratio for the splittable CFLP with non-uniform capacities decreased to $5.83+\epsilon$ [10] and with uniform capacities decreased to 3 [1]. It should be noted that all the known LP relaxations for both the splittable and unsplittable versions have super-constant integrality gap in the general case of the problems.

1.2 The Main Results and Techniques

Recall that given an instance (F, C) of the UCFLP with opening costs f_i, demands d_j, and connection costs c_{ij}, a solution is a subset I of facilities to open along with assignment function $\phi : C \to I$. We use $c_f(\phi)$ to denote the facility cost and $c_s(\phi)$ to denote the service cost; thus $c(\phi) = c_f(\phi) + c_s(\phi)$ will be the total cost. Since all capacities are uniform, by a simple scaling, we can assume that all of them are 1 and all the client demands are at most 1. As we explained before, we are interested in $(O(1), \beta)$-approximation algorithms for some $\beta < 2$. We define a restricted version of the problem and show that finding a good approximation algorithm for this restricted version would imply a good approximation for the general version.

Definition 1. *An ϵ-restricted UCFLP, denoted by RUCFLP(ϵ), instance is an instance of the UCFLP in which each demand has size more than ϵ, i.e., $\epsilon < d_j \leq 1$ for all $j \in C$.*

The following theorem establishes the reduction from the general instances of the UCFLP to the restricted version. Here, the general idea is that if we assign the large clients oblivious to small clients, we can fractionally assign the small clients without paying too much. We use the maximum-flow minimum-cut theorem to show this. Then we can round this fractional assignment of small clients with the GAP rounding technique [8].

Theorem 1. *If $\mathcal{A}$ is an $(\alpha(\epsilon), \beta(\epsilon))$-approximation algorithm for the RUCFLP(ϵ) with running time $\tau(\mathcal{A})$ then there is a $(g(\epsilon, \alpha(\epsilon)), \max\{\beta(\epsilon), 1+\epsilon\})$-approximation algorithm for the UCFLP whose running time is polynomial in $\tau(\mathcal{A})$ and the instance size, where $g(\epsilon, \alpha(\epsilon))$ is a function of ϵ and $\alpha(\epsilon)$, and is linear in $\alpha(\epsilon)$.*

Corollary 1. *For any constant $\epsilon > 0$, an $(\alpha(\epsilon), 1 + \epsilon)$-approximation algorithm for the RUCFLP(ϵ) yields an $(O(\alpha(\epsilon)), 1 + \epsilon)$-approximation for the UCFLP. Particularly, when $\alpha(\epsilon)$ is a constant, we have a constant approximation for the UCFLP with a $(1 + \epsilon)$ violation of capacities in polynomial time.*

This reduction shows it is sufficient to consider large clients only, which may open the possibility of designing algorithms using some of the techniques used in the bin packing type problems. We believe that one can find an $(O(1), 1 + \epsilon)$-approximation algorithm for the RUCFLP(ϵ). If one finds such an algorithm, the above corollary shows that we have an $(O(1), (1 + \epsilon))$-approximation for the UCFLP. As an evidence for this, we find approximation algorithms for the RUCFLP(1/2) and the RUCFLP(1/3). For the RUCFLP(1/2), we present an exact algorithm and for the RUCFLP(1/3), we present a $(21, 1)$-approximation algorithm. These, together with Theorem 1 imply:

Theorem 2. *There is a polynomial time $(10.173, 3/2)$-approximation algorithm for the UCFLP.*

Theorem 3. *There is a polynomial time $(30.432, 4/3)$-approximation algorithm for the UCFLP.*

Finally, we give a QPTAS for Euclidean metrics. Here, we employ a dynamic programming technique and combine the shifted quad-tree dissection of Arora [2], some ideas from [3], and some new ideas to design a dynamic programming.

Theorem 4. *There exists a $(1+\epsilon, 1+\epsilon)$-approximation algorithm for the Euclidean UCFLP in $\mathbb{R}^2$ with running time in quasi-polynomial for any constant $\epsilon > 0$.*

Although this theorem is presented for $\mathbb{R}^2$, it can be generalized to $\mathbb{R}^d$ for constant $d > 2$. Due to lack of space, we defer the proof of Theorem 4 to the full version.

The rest of this paper is organized as follows. In Section 2, we prove Theorem 1. Next, we present approximation algorithms for the RUCFLP(1/2) and RUCFLP(1/3), which also prove weaker versions of Theorems 2 and 3 (see the full version for improved ratios). Finally, in Section 4, we conclude the paper.

2 Reduction to the Restricted UCFLP

In this section, we prove Theorem 1. Let $L = \{j \in C : d_j > \epsilon\}$ be the set of large clients and $S = C \backslash L$ be the set of small clients[1]. We call two assignment $\phi_1 : C_1 \rightarrow F_1$ and $\phi_2 : C_2 \rightarrow F_2$ consistent if $\phi_1(j) = \phi_2(j)$ for all $j \in C_1 \cap C_2$. The high level idea of the algorithm (Algorithm 1) is as follows. We first ignore the small clients and solve the problem restricted to only the large clients by running algorithm $\mathcal{A}$ of Theorem 1. We can show that given a good assignment of large clients, there exists a good assignment of all the clients (large and small) which is consistent with this assignment of large clients, i.e. a solution which assigns the large clients the same way that $\mathcal{A}$ does, whose cost is not far from the optimum cost. More specifically, we show there is a *fractional* (i.e. splittable) assignment of small clients that together with the assignment of large clients obtained from $\mathcal{A}$ gives an approximately good solution. Having this property, we try to find a fractional assignment of small clients. To assign the small clients, we update the capacities and the opening costs of facilities with respect to the assignment of large clients (according to the solution of $\mathcal{A}$). Then, we fractionally assign small clients and round this fractional assignment at the cost of violating the capacities within factor $1+\epsilon$.

First, we formally prove the property that given assignment of large clients, there is a feasible *fractional* assignment of small clients with an acceptable cost. Note that we do not open facilities fractionally and a fractional assignment of demands of (small) clients is essentially equivalent to splitting their demands between multiple open facilities. We should point out that the proof of this property is only an existential result and we do not actually find the assignment in the proof. We only use this lemma to bound the cost of our solution. Let OPT be an optimum solution which opens set I^* of facilities and with assignment of clients $\phi^* : C \rightarrow I^*$. We use $\phi^*_L : L \rightarrow I^*$ and $\phi^*_S : S \rightarrow I^*$ to denote the restriction of ϕ^* to large and small clients, respectively. Here, $\phi^{-1}(i)$ is the

[1] We should point out that the definitions of L and S are with respect to a given parameter ϵ. Since throughout the following sections, this parameter is the same for all statements, in the interest of brevity, we use this notation instead of $L(\epsilon)$ and $S(\epsilon)$.

Algorithm 1 Algorithm for the UCFLP by reduction to the RUCFLP(ϵ)

```
Require: An instance of UCFLP, an $\epsilon > 0$, and the algorithm $\mathcal{A}$ for the RUCFLP($\epsilon$)
Ensure: A subset $I \subseteq F$ of facilities to open and an assignment of clients $\phi : C \to I$
1: Let $L = \{j \in C : d_j > \epsilon\}$ and $S = C \backslash L$. Assign the clients in $L$ by running $\mathcal{A}$. Let
   $I_L$ be the opened facilities and $\phi_L : L \to I_L$ be the assignment found by $\mathcal{A}$.
2: For $i \in I_L$, set $f_i = 0$, and set $u'_i = \max\{0, 1 - \sum_{j \in \phi_L^{-1}(i)} d_j\}$ be the new capacity
   of facility $i$. Assign the clients in $S$ with respect to updated opening costs and
   capacities by an approximation algorithm for the splittable CFLP with non-uniform
   capacities. Let $I_S$ be the new set of opened facilities and $\phi'_S : S \to I'_S$ be the
   assignment function, where $I'_S \subseteq I_S \cup I_L$.
3: Round the splittable assignment $\phi'_S$ using algorithm of [8] to find an unsplittable
   assignment $\phi_S : S \to I'_S$.
4: Let $I = I'_S \cup I_L$ and define $\phi : C \to I$ as $\phi(j) = \phi_S(j)$ if $j \in S$ and otherwise
   $\phi(j) = \phi_L(j)$. Return $\phi$ and $I$.
```

set of clients assigned to facility i by the assignment ϕ and for a $F' \subseteq F$, $\phi^{-1}(F') = \cup_{i \in F'} \phi^{-1}(i)$.

Lemma 1. *Suppose I_L is a set of open facilities and $\phi_L : L \to I_L$ is an arbitrary (not necessarily capacity respecting) assignment of large clients. Given the assignment ϕ_L, there exists a feasible fractional assignment of small clients, $\phi''_S : S \to I''_S$ such that $c_s(\phi''_S) \leq c_s(\phi^*) + c_s(\phi_L)$ and $c_f(\phi''_S) \leq c_f(\phi^*)$.*

Proof. Let $u'_i = \max\{0, 1 - \sum_{j \in \phi_L^{-1}(i)} d_j\}$, i.e., the amount of capacity left for facility i after the assignment of large clients based on ϕ_L. We assume we open all the open facilities in the optimum solution, I^* (if not already open in I_L). Let $I''_S = I_L \cup I^*$. To show the existence of ϕ''_S, first we move the demands of small clients to the facilities in I^* based on ϕ^*_S and we pay $c_s(\phi^*_S)$ for this. So now the demands of small clients are located at facilities in I^*. However, a facility $i \in I''_S$ has only u'_i residual capacity left (after commiting parts of its capacity for the large clients assigned to it by ϕ_L) and this capacity may not be enough to serve the demands of small clients moved to that location. In order to rectify this, we will fractionally redistribute the demands of these small clients between facilities (in I''_S) in such a way that we do not violate capacities u'_i. In this redistribution, we only use the edges used in ϕ_L or ϕ^*_L and if an edge is used to assign large client j to facility i (in ϕ_L or ϕ^*_L), we move at most d_j units of demands of small clients along this edge. Therefore, we pay at most $c_s(\phi_L) + c_s(\phi^*_L)$ in this redistribution. Thus, by the Triangle Inequality, the connection cost of the fractional assignment of small clients obtained at the end is bounded by $c_s(\phi^*_S) + c_s(\phi^*_L) + c_s(\phi_L) = c_s(\phi^*) + c_s(\phi_L)$. Since we only open facilities in the optimum solution (on top of what is already open in I_L) the extra facility cost (for assignment ϕ''_S) is bounded by the facility cost of the optimum.

This process of moving the small client demands can be alternatively thought in the following way. We start from the optimum assignment ϕ^* and change the assignment of large clients to get an assignment identical to ϕ_L for those in

L. Specifically, we change the assignment of a large client j from $i' = \phi^*(j)$ to $i = \phi_L(j)$. This switch increases the amount of demands served at i by d_j and decreases the amount of demand served at i' by d_j. After doing all these switches we might have more demand at some facilities than their capacities, while the total demands assigned to some facilities might be less than 1. To resolve this problem, we try to redistribute (fractionally) the demands of small clients so that there is no capacity violation and we use the max-flow min-cut theorem to show that this redistribution is possible. The details of this part appear in the full version. □

Proof of Theorem 1. Since the cost of the optimum solution for the instance consisting of only the large clients is clearly no more than that of the original instance, after Step 1 of Algorithm 1, we have an assignment ϕ_L such that $c(\phi_L) \leq \alpha(\epsilon)c(\phi^*_L)$ and it violates the capacities by a factor of at most $\beta(\epsilon)$. By Lemma 1, given ϕ_L, there is a feasible fractional assignment ϕ''_S for small clients such that $c_s(\phi''_S) \leq c_s(\phi^*) + c_s(\phi_L)$ and $c_f(\phi''_S) \leq c_f(\phi^*)$.

In Step 2, consider the instance of the splittable CFLP consisting of the small clients and the residual facility opening costs and capacities as defined. We use an approximation algorithm for the splittable CFLP to find an approximate splittable (i.e. fractional) assignment ϕ'_S for small clients. Suppose that the approximation algorithm used for the splittable CFLP has separate factors λ_{ss}, λ_{sf}, λ_{fs}, λ_{ff} such that it returns an assignment with service cost at most $\lambda_{ss}c_s(\tilde{\phi}_S) + \lambda_{sf}c_f(\tilde{\phi}_S)$ and with opening cost $\lambda_{fs}c_s(\tilde{\phi}_S) + \lambda_{ff}c_f(\tilde{\phi}_S)$ for any feasible solution $\tilde{\phi}_S$. Therefore, using Lemma 1:

$$c_s(\phi'_S) \leq \lambda_{ss}c_s(\phi''_S) + \lambda_{sf}c_f(\phi''_S), \tag{1}$$

and

$$c_f(\phi'_S) \leq \lambda_{fs}c_s(\phi''_S) + \lambda_{ff}c_f(\phi''_S). \tag{2}$$

The current best approximation for the splittable CFLP is due to Zhang *et al.* [10] with parameters $\lambda_{ss} = 1$, $\lambda_{sf} = 1$, $\lambda_{fs} = 4$, and $\lambda_{ff} = 5$.

In Step 3, we round the splittable assignment ϕ'_S using the algorithm of Shmoys and Tardos [8] for the Generalized Assignment Problem (GAP) to find an integer assignment ϕ_S. The GAP is a scheduling problem which has similarities to the UCFLP. In the GAP, we have a collection of jobs J and a set M of machines. Each job must be assigned to exactly one machine in M. If job $j \in J$ is assigned to machine $i \in M$, then it requires p_{ij} units of processing and incurs a cost r_{ij}. Each machine $i \in M$ can be assigned jobs of total processing time at most P_i. We want to find an assignment of jobs to machines to minimize the total assignment cost. We should point out that r_{ij} values do not necessarily satisfy the triangle inequality. Shmoys and Tardos [8] show that a feasible fractional solution of the GAP can be rounded, in polynomial time, to an integer solution with the same cost that violates processing time limit P_i within additive factor $\max_{j\in J} p_{ij}$; in worst case this can be a factor 2. We can model (view) the unsplittable capacitated facility location problem as an instance of the GAP in the following sense: jobs are clients, machines are facilities, $p_{ij} = d_j$

for all i, $r_{ij} = d_j \cdot c_{ij}$ for all i and j, and $P_i = 1$, and all facilities are already open (machines are available). Therefore, if we have a fractional assignment of clients to facilities (i.e. a splittable assignment), ϕ'_S, then using the rounding algorithm of [8], we can round ϕ'_S to ϕ_S without increasing the connection cost, i.e. $c_s(\phi_S) = c_s(\phi'_S)$, such that the capacity constraints are violated by at most an additive factor of $\max_{j\in S} d_j$. Since all the jobs in S have demand at most ϵ, the capacity constraints are violated by at most a factor of $1+\epsilon$.

After combining ϕ_S and ϕ_L in Step 4, the violation of capacities is within a factor of at most $\max\{\beta(\epsilon), (1+\epsilon)\}$, because the facilities with violated capacities in Step 1 will be removed in Step 2 and will not be used in Step 3. So it only remains to bound the cost of this assignment:

$$\begin{aligned}
c_s(\phi_S) &= c_s(\phi'_S) && \text{by rounding of [8]}\\
&\le \lambda_{ss} c_s(\phi''_S) + \lambda_{sf} c_f(\phi''_s) && \text{by Equation (1)}\\
&\le \lambda_{ss}(c_s(\phi^*) + c_s(\phi_L)) + \lambda_{sf} c_f(\phi^*), && \text{by Lemma 1}\\
c_f(\phi_S) &\le (1+\epsilon) c_f(\phi'_S) && \text{by rounding of [8]}\\
&\le (1+\epsilon)\lambda_{fs} c_s(\phi''_S) + (1+\epsilon)\lambda_{ff} c_f(\phi''_S) && \text{by Equation (2)}\\
&\le (1+\epsilon)\lambda_{fs}(c_s(\phi^*) + c_s(\phi_L)) + (1+\epsilon)\lambda_{ff} c_f(\phi^*). && \text{by Lemma 1}
\end{aligned}$$

Therefore:

$$\begin{aligned}
c(\phi) &= c(\phi_S) + c(\phi_L)\\
&= c_s(\phi_S) + c_f(\phi_S) + c_s(\phi_L) + c_f(\phi_L)\\
&\le h_1(\epsilon) c_s(\phi^*) + h_2(\epsilon) c_f(\phi^*) + (h_1(\epsilon)+1) c_s(\phi_L) + c_f(\phi_L), \qquad (3)
\end{aligned}$$

where $h_1(\epsilon) = \lambda_{ss} + (1+\epsilon)\lambda_{fs}$ and $h_2(\epsilon) = \lambda_{sf} + (1+\epsilon)\lambda_{ff}$. Since $h_1(\epsilon) \ge 0$ for any $\epsilon > 0$: $(h_1(\epsilon)+1)c_s(\phi_L) + c_f(\phi_L) \le (h_1(\epsilon)+1)c(\phi_L) \le \alpha(\epsilon)(h_1(\epsilon)+1)c(\phi^*_L) \le \alpha(\epsilon)(h_1(\epsilon)+1)c(\phi^*)$. Combining this with Inequality (3), we obtain that the cost of ϕ is within factor:

$$g(\epsilon, \alpha(\epsilon)) = \max(h_1(\epsilon), h_2(\epsilon)) + \alpha(\epsilon)(h_1(\epsilon)+1) \qquad (4)$$

of the optimum. □

3 The RUCFLP($\frac{1}{2}$) and RUCFLP($\frac{1}{3}$)

In this section, we give two approximation algorithms for the RUCFLP($\frac{1}{2}$) and RUCFLP($\frac{1}{3}$). Combined with Theorem 1 (and using Algorithm 1) these imply two approximation algorithms for the UCFLP. We start with the simpler of the two, namely the RUCFLP($\frac{1}{2}$).

Theorem 5. *There is a polynomial time exact algorithm for the RUCFLP($\frac{1}{2}$).*

Proof. Consider an optimal solution for a given instance of this problem with value OPT_L. Because $d_j > \frac{1}{2}$ for all $j \in C$, each facility can serve at most one client in the optimal solution. Therefore, the optimal assignment function,

ϕ_L^*, induces a matching $M = \{j\phi_L^*(j) : j \in C\}$. Let $w_{ij} = c_{ij}.d_j + f_i$ and let $w(H) = \sum_{e \in H} w_e$ for any subset of edges $H \subseteq E$. It follows that $w(M) = \mathrm{OPT}_L$.

Let M^* be a minimum weight perfect matching with respect to weights w_{ij}. Clearly, $w(M^*) \leq w(M) = \mathrm{OPT}_L$. In addition, M^* induces a feasible assignment of clients to facilities with cost $w(M^*)$. Thus, M^* induces an optimal solution for the RUCFLP($\frac{1}{2}$). Since we can find a minimum weight perfect matching in polynomial time, there is an exact algorithm for the RUCFLP($\frac{1}{2}$). □

Corollary 2. *There is a polynomial time* $(16.5, 3/2)$*-approximation algorithm for the UCFL problem.*

Proof. We run Algorithm 1, where we use the algorithm of Theorem 5 in the first step. Substituting $\alpha(\epsilon) = 1$ and $\epsilon = 1/2$, we have $h_1(\frac{1}{2}) = 7$, $h_2(\frac{1}{2}) = 8.5$, and $g(\epsilon, \alpha(\epsilon)) = 16.5$. Since $\beta(\epsilon) = 1$, the overall ratio is $(16.5, 3/2)$. □

The algorithm for the RUCFLP($\frac{1}{3}$) is more involved. First, we show how finding an approximation algorithm for the RUCFLP(ϵ) with zero facility opening costs leads to an approximation algorithm for the general RUCFLP(ϵ). Then, we give an approximation algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs.

Lemma 2. *Given an algorithm $\mathcal{A}'$ for the RUCFLP(ϵ) with zero facility opening costs having approximation factor $(\alpha'(\epsilon), \beta(\epsilon))$, we can find a $(\alpha'(\epsilon)\frac{1}{\epsilon}, \beta(\epsilon))$-approximation algorithm $\mathcal{A}$ for the general RUCFLP(ϵ).*

Proof. Define a new connection cost $c'_{ij} = c_{ij} + f_i$ and opening cost $f'_i = 0$ for all $i \in F$ and $j \in C$. Note that the new cost function is still metric. Then, we run $\mathcal{A}'$ on this new modified instance and let the solution returned by it be assignment ϕ_L. We use ϕ_L to assign the clients for the original instance and we claim this is a $(\alpha'(\epsilon)\frac{1}{\epsilon}, \beta(\epsilon))$-approximation. The proof of this appears in the full version. □

Now, we present a $(7, 1)$-approximation algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs (see Algorithm 2), which coupled with Lemma 2 yields a $(21, 1)$-approximation algorithm for the RUCFLP($\frac{1}{3}$).

Theorem 6. *There is a* $(7, 1)$*-approximation algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs.*

Proof. Note that all the clients in the given instance have size $> \frac{1}{3}$. We break them into two groups: $L' = \{j \in C : d_j > \frac{1}{2}\}$ and $L'' = C \backslash L'$ are those which have size in $(\frac{1}{3}, \frac{1}{2}]$. In this proof (and of Lemma 3), we call clients in L', *huge* clients and those in L'', *moderately-large* clients. The algorithm assigns the huge clients by running a minimum weight perfect matching algorithm with edge weights $w_{ij} = d_j c_{ij}$. Let $I_{L'}$ be the opened facilities and $\phi_{L'} : L' \rightarrow I_{L'}$ be the assignment function. For moderately-large clients (i.e. those in L''), we define a flow-network H and show that minimum cost maximum flows in H correspond to minimum cost feasible assignment of clients in L'' to facilities (given the assignment $\phi_{L'}$).

Directed network H has node set $X \cup Y \cup \{s, t\}$ where there is a node $x_j \in X$ for every client $j \in L''$ and a node $y_i \in Y$ for every facility $i \in F$; s is the

Algorithm 2 Algorithm for solving the RUCFLP($\frac{1}{3}$) with zero opening costs

```
Require: An instance of the RUCFLP($\frac{1}{3}$) with zero opening costs
Ensure: A subset $I \subseteq F$ and a function $\phi : C \rightarrow I$
1: Let $L' = \{j \in C : d_j > \frac{1}{2}\}$ and $L'' = C \backslash L'$. Assign the clients in $L'$ by running a minimum weight maximum matching algorithm that saturates $L'$ with edge weights $w_{ij} = d_j c_{ij}$. Let $I_{L'}$ be the opened facilities and $\phi_{L'} : L' \rightarrow I_{L'}$ be the assignment function.
2: Build the flow network $H$ as described in Theorem 6.
3: Find a minimum cost maximum flow in $H$. If the value of the flow is smaller than $|L''|$ then return "Infeasible". Else, let $I_{L''}$ be the subset of facilities in $F \backslash I_{L'}$ whose corresponding nodes in $Y$ (in $H$) have non-zero flow through them and $\phi_{L''}$ be the assignment function defined as: if there is a unit flow from $x_j$ to $y_i$ in $H$ then $\phi_{L''}(j) = i$.
4: Let $I = I_{L''} \cup I_{L'}$. Combine $\phi_{L''}$ and $\phi_{L'}$ to form assignment function $\phi_L : C \rightarrow I$ where $\phi(j) = \phi_{L''}(j)$ if $j \in L''$, otherwise $\phi(j) = \phi_{L'}(j)$. Return $\phi$ and $I$.
```

source and t is the sink. The source is connected to each node $x_j \in X$, and all $y_i \in Y$ are connected to the sink. Each $x_j \in X$ is connected to a node $y_i \in Y$ if either: the corresponding facility i is in $F \backslash I_{L'}$, i.e. unopened yet, or i is in $I_{L'}$ and the remaining capacity of i is enough to serve the demand of client j. Set the capacity of the edges between the source and the nodes in X to 1, set the capacity of the edges between X and Y to 1, set the capacity of the edges between the nodes $y_i \in Y$ whose corresponding facility i is unopened (i.e. not in $I_{L'}$) and the sink to 2, and set the capacity of the edges between the nodes $y_i \in Y$ whose corresponding facility is in $I_{L'}$ and the sink to 1. The cost of an edge connecting $x_j y_i$ is $d_j \cdot c_{ij}$ and all the other costs are 0. Algorithm 2 summarizes the algorithm for the RUCFLP($\frac{1}{3}$) with zero opening costs.

Let ϕ^*_L be an optimal assignment for the given instance of the RUCFLP($\frac{1}{3}$) with cost OPT_L. We use the following lemma (whose proof appears in the full version):

Lemma 3. *There exists an assignment ϕ' of clients consistent with $\phi_{L'}$ with cost at most $7\mathrm{OPT}_L$ where OPT_L is the cost of an optimum assignment ϕ^*_L for the given instance of the RUCFLP($\frac{1}{3}$).*

Below we prove that in Steps 2 and 3 the algorithm finds the best possible feasible assignment of clients in L'' (given $\phi_{L'}$). Therefore, the cost of ϕ formed in Step 4 is at most $c(\phi')$ and hence, is at most $7\mathrm{OPT}_L$.

Since for any $j \in L''$: $\frac{1}{3} < d_j \leq \frac{1}{2}$, each unopened facility after Step 1 can serve any two clients of L'' (and no more than two). This fact is reflected in that we connect all the nodes in X (corresponding to moderately-large clients) to the nodes in Y corresponding to unopened facilities $F \backslash I_{L'}$ and we set the capacity of the edges between those nodes in Y and the sink to 2. In addition, each facility in $I_{L'}$ can serve at most one moderately-large client, because more than $\frac{1}{2}$ of its capacity is already used by a huge client; accordingly we set the capacity of the edges from those nodes in Y to the sink to 1 and we only connect to them the nodes of X whose corresponding client can be served by them. Considering these two simple facts (proof in the full version):

Lemma 4. *The maximum flow in H has value $|L''|$ if and only if the given instance is feasible and there is a one to one correspondence between maximum flows in H and feasible assignment of moderately-large clients (i.e. in L'') given $\phi_{L'}$. Furthermore, a pair of corresponding maximum flow in H and assignment of clients of L'' to F have the same cost.*

Therefore, the assignment $\phi_{L''}$ obtained from a minimum cost maximum flow in H has the minimum cost among the assignments consistent with $\phi_{L'}$. This together with Lemma 3 implies that ϕ_B as defined has cost at most $7\mathrm{OPT}_L$. □

Combining Lemma 2 and Theorem 6:

Corollary 3. *There is a polynomial time $(21, 1)$-approximation algorithm for the RUCFLP($\frac{1}{3}$).*

Corollary 4. *There is a (161.667,4/3)-approximation algorithm for the UCFL problem.*

Proof. We run Algorithm 1, where we use the algorithm of Corollary 3 for $\mathcal{A}$. That is, we first run the $(7, 1)$-approximation algorithm of Theorem 6 as algorithm $\mathcal{A}'$ in Lemma 2 to obtain $\mathcal{A}$ with $\alpha(\epsilon) = 21$ and $\epsilon = 1/3$. Thus $h_1(\frac{1}{3}) = 19/3$, $h_2(\frac{1}{3}) = 23/3$, and $g(\epsilon, \alpha(\epsilon)) = 23/3 + 21(22/3) < 161.667$. Since $\beta(\epsilon) = 1$, the overall ratio is $(161.667, 4/3)$. □

With a more careful analysis and a simple scaling to balance the bi-factors of connection and facility costs, we can bring down the factors of our algorithms (see the full version). This will imply the improved ratios in Theorem 2 and 3.

Notice that we solved the RUCFLP($\frac{1}{3}$) and the the RUCFLP($\frac{1}{2}$) without violation of capacities, but this is not possible for smaller values of ϵ as shown below (see the full version for the proof).

Theorem 7. *The RUCFLP(ϵ) does not admit any $(\alpha(\epsilon), 1)$-approximation algorithm for $\epsilon < \frac{1}{3}$ unless $P = NP$.*

It should be noted that to find an algorithm for the UCFLP that violates capacities within factor $1 + \epsilon$, we do not need to find an algorithm that does not violate capacities in the RUCFLP(ϵ). Even if we violate the capacities within factor $1 + \epsilon$ in the RUCFLP(ϵ), by Theorem 1 we can get an algorithm for the UCFLP that violates the capacities within factor $1+\epsilon$. We think it is possible to find an $(\alpha(\epsilon), 1 + \epsilon)$-approximation for the RUCFLP(ϵ) for any constant $\epsilon > 0$. This, together with Theorem 1 would imply an $(f(\epsilon), 1 + \epsilon)$-approximation for the UCFLP, for any constant $\epsilon > 0$.

4 Discussion

We presented a reduction from the UCFLP to a restricted version in which all demand values are large (i.e. $> \epsilon$) and presented two algorithms for the case of

$\epsilon = \frac{1}{2}$ and $\frac{1}{3}$. These implied two constant factor approximation algorithms for the UCFLP with capacity bounds within factor 3/2 and 4/3. We believe similar results can be found with capacity violations bounded within factor $1+\epsilon$ for any $\epsilon > 0$. We also showed that at a loss of factor $1/\epsilon$, we can ignore the opening cost of facilities, and that if there is an $(\alpha(\epsilon), 1+\epsilon)$-approximation for these instances then there is an $(\alpha'(\epsilon), 1+\epsilon)$-approximation for the general case. We believe that it should be possible to design constant factor (perhaps depending on ϵ) approximation for RUCFLP(ϵ) with zero opening costs with a violation of at most $1+\epsilon$ on capacities.

Acknowledgements. Part of this work was done while the second author was on sabbatical at Toyota Tech. Inst. at Chicago (TTIC). He would like to thank TTIC for hosting him.

References

1. Aggarwal, A., Anand, L., Bansal, M., Garg, N., Gupta, N., Gupta, S., Jain, S.: A 3-Approximation for Facility Location with Uniform Capacities. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 149–162. Springer, Heidelberg (2010)
2. Arora, S.: Polynomial time approximation schemes for euclidean tsp and other geometric problems. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science. pp. 2–12 (1996)
3. Bateni, M., Hajiaghayi, M.: Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 805–814 (2009)
4. Guha, S., Khuller, S.: Greedy strikes back: improved facility location algorithms. In: SODA 1998: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 649–657 (1998)
5. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1998, pp. 1–10 (1998)
6. Li, S.: A 1.488 Approximation Algorithm for the Uncapacitated Facility Location Problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 77–88. Springer, Heidelberg (2011)
7. Mahdian, M., Ye, Y., Zhang, J.: A 2-Approximation Algorithm for the Soft-Capacitated Facility Location Problem. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 129–140. Springer, Heidelberg (2003)
8. Shmoys, D., Tardos, E.: An approximation algorithm for the generalized assignment problem. Mathematical Programming 62(3), 461–474 (1993)
9. Shmoys, D., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 265–274 (1997)
10. Zhang, J., Chen, B., Ye, Y.: A Multi-exchange Local Search Algorithm for the Capacitated Facility Location Problem. In: Bienstock, D., Nemhauser, G.L. (eds.) IPCO 2004. LNCS, vol. 3064, pp. 219–233. Springer, Heidelberg (2004)

Non-preemptive Speed Scaling

Antonios Antoniadis and Chien-Chung Huang

Department of Computer Science, Humboldt-Universität zu Berlin,
Unter den Linden 6, 10099 Berlin

Abstract. We consider the following variant of the speed scaling problem introduced by Yao, Demers, and Shenker. We are given a set of jobs and we have a variable-speed processor to process them. The higher the processor speed, the higher the energy consumption. Each job is associated with its own release time, deadline, and processing volume. The objective is to find a feasible schedule that minimizes the energy consumption. Moreover, no preemption of jobs is allowed.

Unlike the preemptive version that is known to be in P, the non-preemptive version of speed scaling is strongly NP-hard. In this work, we present a constant factor approximation algorithm for it. The main technical idea is to transform the problem into the unrelated machine scheduling problem with L_p-norm objective.

1 Introduction

The *speed scaling* problem was introduced by Yao, Demers and Shenker [16] in 1995. The input is a set $\mathcal{J}$ of jobs. Each job $j \in \mathcal{J}$ is associated with a release time r_j, a deadline d_j and a volume v_j. W.l.o.g., we assume that release times and deadlines are given as integers. We have a variable-speed processor, which is associated with a *power function* $P(s) = s^\alpha$ with $\alpha > 1$. Assuming that job $j \in \mathcal{J}$ is always processed at a speed of s_j, then j requires v_j/s_j time to be completed. The energy consumption of the processor is power integrated over time. A schedule is said to be feasible if the volume of each job is completely processed not before its release time but before its deadline. The goal is to find a feasible schedule that minimizes the total energy consumption.

Given our growing awareness of the environment, speed scaling is a natural problem to study. Practically, many modern microprocessors, in order to be more energy-efficient, have the built-in capability to vary their speed. It is desirable to have good scheduling policies in order to reduce the energy consumption in such computing environments. Furthermore this can help prolonging battery lives in mobile devices.

The original model of Yao et al. assumes that jobs can be preempted. That is, the execution of a job may be paused and resumed at a later point in time. The *non-preemptive* version of this problem, where a job, once started, must be processed until its completion, has not been studied so far. We call the problem *non-preemptive speed scaling.* The assumption that preemption is disallowed is a natural one to make, since preemption is known to cause significant overhead in

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 249–260, 2012.

practice. The fact that some theoretically good scheduling algorithms perform relatively bad in practice has been attributed to this overhead [10,11,13]. Furthermore, since the preemption of a job may require storing the state of both the system and the processing of the job, the implementation of preemption can become particularly hard in some settings. Some researchers try to cope with the above problems by either limiting or completely avoiding preemption (see [4,5] for some examples).

Unlike the preemptive speed scaling, for which Yao et al. gave a simple and elegant polynomial time algorithm, the non-preemptive speed scaling problem is NP-hard. To see this, consider the following reduction from the *partition* problem [12]. In the partition problem, a set of integers $N = \{n_1, n_2, \cdots\}$ is given. The question is whether the set N can be partitioned into two disjoint subsets, so that the sum of one subset is equal to the sum of the other subset. We create a job j with release time 1 and deadline 2. Its volume v_j is $\sum_i n_i$. Further, for each integer $n_t \in N$, create a job of volume v_t with release time 0 and deadline 3. Due to the fact that the power function $P(s) = s^\alpha$ is strictly convex, it is easy to see that the original instance allows a partition if and only if the optimal schedule uses the same speed in the intervals $[0, 1)$ and $[2, 3)$. By a straightforward generalization of the idea, we can reduce from the *3-Partition* problem [12] to show that non-preemptive speed scaling is strongly NP-hard.

We note that even though the general case of the problem is NP-hard, there is a natural special case that is in P: when the given instance has *agreeable deadlines.* An instance has agreeable deadlines if given any two jobs $j, j' \in \mathcal{J}$ with $r_j < r_{j'}$, then $d_j \leq d_{j'}$. In words, earlier-released jobs also have earlier deadlines. For this special case, the original algorithm of Yao et al. can be applied by the observation that the schedule returned does not make use of preemption. However, to deal with the general case, novel algorithmic ideas are required.

Our Contribution and Technique. We develop a constant-factor approximation algorithm for the problem of non-preemptive speed scaling.

Theorem 1. *There exists a polynomial time algorithm that achieves a $2^{5\alpha-4}$-approximation for the non-preemptive speed scaling problem. For the special case of* laminar instances*, there exists a polynomial time algorithm that achieves a $2^{4\alpha-3}$-approximation.*

An instance is said to be a laminar instance if for any two jobs $j, j' \in \mathcal{J}$, either $[r_j, d_j) \subseteq [r_{j'}, d_{j'})$, $[r_{j'}, d_{j'}) \subseteq [r_j, d_j)$, or $[r_j, d_j) \cap [r_{j'}, d_{j'}) = \emptyset$. Laminar instances are of interest for two reasons. First, they form a natural special class of instances. As reported in [14], when the jobs are created by recursive calls in a program, the resulting instance is laminar. Secondly, and more importantly, we observe that a laminar instance is in a sense the "opposite" of an instance with agreeable deadlines: for any two jobs that have overlapping intervals, the one with earlier release time must have a latter deadline. As the agreeable version can be solved in polynomial time, this "opposite" version highlights the difficulty of the non-preemptive speed scaling problem. (Observe that the reductions we used earlier from partition/3-partition are laminar instances as well).

In Section 3, we show how to achieve a $2^{4\alpha-3}$ approximation for laminar instances. The rough idea is to use a series of transformations, which involve the "chopping up" of the intervals of the jobs, so that we can reduce a laminar instance into an instance of unrelated machine scheduling with the L_α-norm objective, which is known to be 2-approximable [2]. The "chopping up" of an interval, roughly speaking, means that we create new instances with the additional constraint that a job has to be processed entirely within a sub-interval. The technical challenge is to bound the growth of the cost throughout the series of transformations.

In Section 4, we develop a sweepline algorithm and use a certain "energy-folding" technique, to reduce a general instance into a laminar instance, at a further cost of $2^{\alpha-1}$ in the approximation factor.

Due to space constraints some of the proofs are omitted.

Previous Work. Even though dynamic speed scaling has been studied extensively over the past years, to the best of our knowledge, non-preemptive speed scaling was not considered before. As already mentioned, the study of the (preemptive) problem was initiated by Yao et al. [16], who have developed a polynomial-time algorithm for the offline problem. Some special cases of the preemptive version of the problem have also been studied. For example, when the given instance is laminar [14] or when the power function is discrete [15].

In [8] and [9] constant-factor approximation algorithms for the problem on fixed-speed processors were given. In [8], more than one processor may be used in order to feasibly schedule the jobs, and the objective is to minimize the number of used processors. In [9] one may skip jobs in order to produce a feasible schedule. The objective here is to maximize the number of scheduled jobs. In [7] a similar problem of scheduling jobs without preemption on a variable-speed processor was studied. However in their model the processor can only operate at discrete speed levels, and there are precedence constraints among the jobs. They show that their problem is NP-hard and give polynomial time approximation schemes for two restricted cases.

So far we only discussed the offline version of the speed scaling problem. The online version has also been extensively studied (but still only for the preemptive version). Yao et al. [16] presented two online algorithms called *Average Rate* and *Optimal Available*. For the first they have proven a competitive ratio of $(2\alpha)^\alpha/2$ whereas the second was analyzed by Bansal, Kimbrel and Pruhs [3] who have shown a tight competitive ratio of α^α.

For a recent literature review on energy efficient algorithms, see [1].

2 Preliminaries

We define a more general version of our problem. Let $\mathcal{J}$ be the set of jobs. Each job $j \in \mathcal{J}$ has a set of disjoint *allowed intervals*, $I_{j1} = [r_{j1}, d_{j1})$, $I_{j2} = [r_{j2}, d_{j2})$, $\cdots$. A schedule $\mathcal{S}$ is *feasible* if each job $j \in \mathcal{J}$ is executed *entirely within* one of its allowed intervals. Note that in the original problem instance $\mathcal{I}$, each job has only one allowed interval. We will transform $\mathcal{I}$ so that a job may have multiple intervals. Let $d_{\max}$ be the latest deadline in the original instance $\mathcal{I}$. Assume without loss of generality, that the earliest release time among all the jobs in $\mathcal{J}$ is 0.

Throughout the article, we implicitly assume that a feasible schedule processes a job using a uniform speed. This assumption is without loss of generality, since the power function $P(s) = s^\alpha$ is strictly convex and using a uniform speed for a given job minimizes its expended energy.

Given a schedule $\mathcal{S}$, let $\mathcal{S}(j) = [b_j, e_j)$ denote the execution interval of job j and $e_j(\mathcal{S})$ the finishing time of job j under schedule $\mathcal{S}$. Furthermore, let $E(\mathcal{S})$ denote the total energy spent by schedule $\mathcal{S}$ and $E(\mathcal{S}, j)$ be the energy used for processing job j under schedule $\mathcal{S}$. The following proposition follows from our assumption that a feasible schedule processes each job at uniform speed.

Proposition 1. *Suppose that schedules $\mathcal{S}$ and $\mathcal{S}'$ process job j with speed s and s' respectively. Assume that $s \leq cs'$ for some $c \geq 1$. Then $E(\mathcal{S}, j) \leq c^{\alpha-1}E(\mathcal{S}', j)$.*

3 Special Case: Laminar Family

In this section we assume that the given instance $\mathcal{I}$ is a laminar instance. Recall that for such an instance, for any two jobs $j, j' \in \mathcal{J}$, either $[r_j, d_j) \subseteq [r_{j'}, d_{j'})$, $[r_{j'}, d_{j'}) \subseteq [r_j, d_j)$, or $[r_j, d_j) \cap [r_{j'}, d_{j'}) = \emptyset$. We can associate the jobs in $\mathcal{I}$ with a tree structure as follows. A job j is a descendant of another job j' in the tree if and only if $[r_j, d_j) \subset [r_{j'}, d_{j'})$. A job j is said to be a *leaf job* if its interval $[r_j, d_j)$ is not a proper superset of the interval of any other job. In case that there are more than one candidate leaf jobs with identical intervals $[r_j, d_j)$, we pick an arbitrary one of them as a leaf job. Let the leaf jobs form a set L. Note that all the jobs in L have distinct deadlines.

Transformation I

Our first step is to partition the entire interval $[0, d_{\max})$ into "zones" using a set of "landmarks."

Definition 1. *Let $\tau_1 < \tau_2 < \cdots < \tau_{|L|}$ be the set of ordered deadlines d_j for the jobs $j \in L$. Furthermore, let $\tau_0 = 0$ and $\tau_{|L|+1} = d_{\max}$. (Note that $\tau_0 < \tau_1$ and $\tau_{|L|} \leq \tau_{|L|+1}$.) The intervals $[\tau_i, \tau_{i+1})$ for all $0 \leq i \leq |L|$ are called* zones. *Throughout the text, we will refer to the τ_i's as* landmarks.

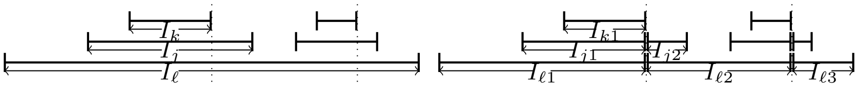
We create a new instance $\mathcal{I}_1$ as follows. Consider a job $j \in \mathcal{J}$ with its allowed interval $I_j = [r_j, d_j)$, such that

$$\tau_{i-1} \leq r_j < \tau_i < \tau_{i+1} < \cdots < \tau_{i+k} < d_j \leq \tau_{i+k+1}.$$

We replace its allowed interval $I_j = [r_j, d_j)$ with a set of allowed intervals $\bigcup_{s=1}^{k+2} I_{js}$, where

$$I_{j1} = [r_j, \tau_i),\ I_{j(k+2)} = [\tau_{i+k}, d_j),\ \text{and } I_{js} = [\tau_{s+i-2}, \tau_{s+i-1}) \text{ for } 2 \le s \le k+1.$$

See Figure 1 for an illustration. We now show that the above partition of jobs' allowed intervals does not increase the cost of an optimal solution by too much.



(a) The original instance $\mathcal{I}$. (b) The modified instance $\mathcal{I}_1$.

Fig. 1. In (a), the two dotted lines are the landmarks, since they are the deadlines of the leaf jobs. In (b), we show how to use the landmarks to divide a job's allowed interval in $\mathcal{I}$ into several allowed intervals in the modified instance $\mathcal{I}_1$.

Lemma 1. *Let $OPT_{\mathcal{I}}$ and $OPT_{\mathcal{I}_1}$ denote the optimal schedules for instances $\mathcal{I}$ and $\mathcal{I}_1$ respectively. Then $E(OPT_{\mathcal{I}_1}) \le 2^{\alpha-1} E(OPT_{\mathcal{I}})$.*

Transformation II

In the second transformation, inside each zone $[\tau_{i-1}, \tau_i)$, we define a set of *sub-landmarks* to further subdivide the zone into a set of *subzones*. The allowed intervals of a job are (possibly) shortened and then further fragmented by these subzones. We now flesh out the details.

Recall that each job in $\mathcal{J}$ has at most one allowed interval in the zone $[\tau_{i-1}, \tau_i)$. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the subset of the jobs that have exactly one allowed interval in this zone. For simplicity, we assume that the allowed interval of a job $j \in \mathcal{J}'$ is also its first allowed interval $I_{j1} = [r_{j1}, d_{j1})$. We divide the jobs in $\mathcal{J}'$ into three groups.

$$\begin{aligned} j \in A &\text{ if } r_{j1} = \tau_{i-1}, d_{j1} < \tau_i; \\ j \in B &\text{ if} \tau_{i-1} < r_{j1}, d_{j1} = \tau_i; \\ j \in C &\text{ if} r_{j1} = \tau_{i-1}, d_{j1} = \tau_i. \end{aligned}$$

See Figure 2(a) for an illustration. Observe that jobs in group C have their allowed intervals span the entire zone; furthermore, by our assumption that the original instance $\mathcal{I}$ is a laminar instance, the allowed intervals of jobs in group A do not overlap with the allowed intervals of jobs in group B.

Lemma 2. *Let $\mathcal{S}$ be a feasible schedule for instance I_1 and let $\mathcal{J}''$ be the jobs of $\mathcal{J}'$ that are processed within $[\tau_{i-1}, \tau_i)$. Then $\mathcal{S}$ can be transformed into a feasible schedule $\mathcal{S}'$ with the following properties:*

- *$\mathcal{S}'$ consumes no more energy than $\mathcal{S}$.*
- *All jobs in $\mathcal{J}''$ are processed within the zone $[\tau_{i-1}, \tau_i)$.*

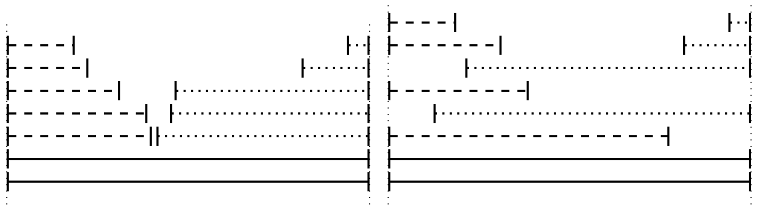


Fig. 2. A zone. The dashed lines represent allowed intervals for jobs of group A, the dotted ones for jobs of group B. Finally, the solid lines represent the allowed intervals for jobs of group C. (a) illustrates a zone after Transformation I. (b) illustrates a zone after Transformation I_1 (see Section 4).

- *$\mathcal{S}'$ executes all the jobs of group $A \cap \mathcal{J}''$ before the jobs of group $C \cap \mathcal{J}''$, which in turn are executed before the jobs of group $B \cap \mathcal{J}''$, in $[\tau_{i-1}, \tau_i)$. Moreover, in $\mathcal{S}'$, the jobs of $A \cap \mathcal{J}''$ are processed according to the earliest deadline first principle and the jobs of $B \cap \mathcal{J}''$ according to the earliest release time first principle.*

Let d_{Al} be the latest deadline for the jobs in group A and r_{Bf} the earliest release time for the jobs in group B. Let $\lambda_A = \tau_{i-1} + 3/4(d_{Al} - \tau_{i-1})$ and $\lambda_B = \tau_i - 3/4(\tau_i - r_{Bf})$ be two sublandmarks and $[\lambda_A, \lambda_B)$ be a subzone (we will define more sublandmarks inside $[\tau_{i-1}, \lambda_A)$ and $[\lambda_B, \tau_i)$ in a moment). Then

$$\lambda_B - \lambda_A = (\lambda_B - r_{Bf}) + (d_{Al} - \lambda_A) + (r_{Bf} - d_{Al}) =$$
$$\frac{(\tau_i - r_{Bf}) + (d_{Al} - \tau_{i-1})}{4} + r_{Bf} - d_{Al} = \frac{\tau_i - \tau_{i-1}}{4} + \frac{3(r_{Bf} - d_{Al})}{4} \geq \frac{\tau_i - \tau_{i-1}}{4}. \tag{1}$$

Therefore, the subzone $[\lambda_A, \lambda_B)$ is of length at least a fourth of the entire zone. Intuitively speaking, in our proof, this subzone is "reserved" for the jobs in group C. Given any schedule for $\mathcal{I}_1$, even if the processing of the jobs in group C spans the entire zone, we can always increase their processing speed by a factor of 4 and process them entirely within the subzone $[\lambda_A, \lambda_B)$.

Let d_{Af} be the earliest deadline in group A and r_{Bl} the latest release time in group C. Also, let x be the largest positive integer so that $\tau_{i-1} + (d_{Af} - \tau_{i-1})2^{x-1} < \lambda_A$ and y to be the largest positive integer so that $\tau_i - (\tau_i - r_{Bl})2^{y-1} > \lambda_B$. We define the following sublandmarks.

$$\begin{aligned}
\lambda_A^k &= \tau_{i-1} + (d_{Af} - \tau_{i-1})2^{k-1}, && \text{for } 1 \leq k \leq x;\\
\lambda_A^{x+1} &= \lambda_A;\\
\lambda_B^{y+1} &= \lambda_B;\\
\lambda_B^k &= \tau_i - (\tau_i - r_{Bl})2^{k-1}, && \text{for } 1 \leq k \leq y.
\end{aligned}$$

Notice that in the case that $\tau_{i-1} + (d_{Af} - \tau_{i-1}) = d_{Af} \geq \lambda_A$, we do not define any sublandmark in $[\tau_{i-1}, \lambda_A)$; similarly for $[\lambda_B, \tau_i)$, if $\tau_i - (\tau_i - r_{Bl}) \geq \lambda_B$. (Also note that it is possible that $d_{Af} = d_{Al}$ or $r_{Bf} = r_{Bl}$). Finally, observe that as we assume that all release times and deadlines are integers, there can be only $O(\lceil \log d_{\max} \rceil)$ sublandmarks in each zone.

The sublandmarks $\{\lambda_A^k\}_{k=1}^x$ and $\{\lambda_B^k\}_{k=1}^y$ are used to partition the intervals $[\tau_{i-1}, \lambda_A)$ and $[\lambda_B, \tau_i)$ into subzones respectively. See Figure 3 for an illustration. It can be observed that the sizes of the subzones in $[\tau_{i-1}, \lambda_A)$ grow geometrically, except the first and the last one; similarly, the sizes of the subzones $[\lambda_B, \tau_i)$, except the first and the last one, decrease geometrically. Roughly speaking, in our proof, these subzones will be "reserved" for jobs in group A and B respectively.

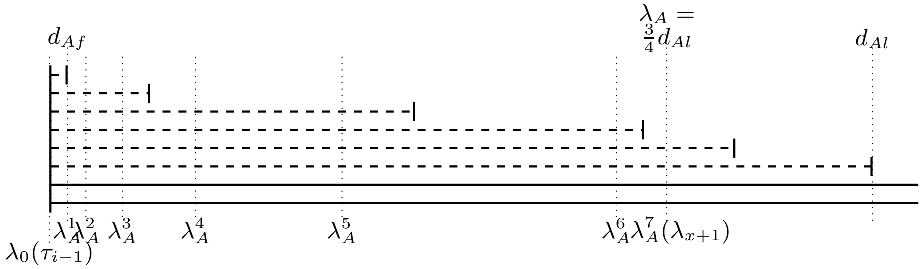

d_{Af}
$\lambda_A = \frac{3}{4} d_{Al}$
d_{Al}
$\lambda_A^1 \lambda_A^2$ λ_A^3
λ_A^4
λ_A^5
$\lambda_A^6 \lambda_A^7 (\lambda_{x+1})$
$\lambda_0(\tau_{i-1})$

Fig. 3. The sublandmarks inside the interval $[\tau_{i-1}, \lambda_A)$

We now use the set of sublandmarks $\{\lambda_A^k\}_{k=1}^x \cup \{\lambda_B^k\}_{k=1}^y \cup \lambda_A \cup \lambda_B$ to partition the allowed intervals of all jobs in $\mathcal{J}'$ in a somewhat similar manner to the previous transformation. Suppose there are totally g sublandmarks in the zone $[\tau_{i-1}, \tau_i)$. Let λ_w denote the w-th sublandmark (in increasing order) for $1 \leq w \leq g$. For convenience, let $\lambda_0 = \tau_{i-1}$ and $\lambda_{g+1} = \tau_i$.

We create a new instance $\mathcal{I}_2$ based on $\mathcal{I}_1$ as follows. For a job $j \in \mathcal{J}'$ with the allowed interval $I_{j1} = [r_{j1}, d_{j1})$, let λ_u be the first sublandmark so that $r_{j1} \leq \lambda_u$ and λ_{u+k} be the last sublandmark so that $\lambda_{u+k} \leq d_{j1}$. Then

$$r_{j1} \leq \lambda_u < \lambda_{u+1} < \cdots < \lambda_{u+k} \leq d_{j1}.$$

We replace its allowed interval $I_{j1} = [r_{j1}, d_{j1})$ with a set of allowed intervals $\bigcup_{s=1}^k I_{j1s}$, where

$$I_{j1s} = [\lambda_{u+s-1}, \lambda_{u+s}), \forall s, 1 \leq s \leq k.$$

Note that by this definition, a subzone $[\lambda_{k-1}, \lambda_k)$ becomes an allowed interval of job j in instance $\mathcal{I}_2$ if and only if I_{j1} spans the *entire* interval $[\lambda_{k-1}, \lambda_k)$. In the case that $r_{j1} < \lambda_u$ (this may happen for jobs in $B \cap \mathcal{J}''$) or $d_{j1} > \lambda_{u+k}$ (which may happen for jobs in $A \cap \mathcal{J}''$), $[r_{j1}, \lambda_u)$ or $[\lambda_{u+k}, d_{j1})$ respectively are not allowed intervals.

Lemma 3. *Let $OPT_{\mathcal{I}_1}$ and $OPT_{\mathcal{I}_2}$ denote the optimal schedules for instances $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. Then $E(OPT_{\mathcal{I}_2}) \leq 4^{\alpha-1} E(OPT_{\mathcal{I}_1})$.*

Proof. We construct a feasible solution $\overline{OPT}_{\mathcal{I}_2}$ for instance $\mathcal{I}_2$ based on the optimal solution $OPT_{\mathcal{I}_1}$ for instance $\mathcal{I}_1$. Let $\mathcal{J}'' \subseteq \mathcal{J}'$ be the subset of jobs that are processed in the zone $[\tau_{i-1}, \tau_i)$ in the schedule $OPT_{\mathcal{I}_1}$. By Lemma 2, we can assume that the jobs in $A \cap \mathcal{J}''$ are processed first, the jobs in $C \cap \mathcal{J}''$ second, followed by the jobs in $B \cap \mathcal{J}''$. We also can assume that jobs in $A \cap \mathcal{J}''$ (resp. $B \cap \mathcal{J}''$) are processed in the order of their increasing deadlines (resp. increasing release times).

Suppose that there exists at least one sublandmark within $[\tau_{i-1}, \lambda_A)$. Recall that we define $\lambda_A^k = \tau_{i-1} + (d_{Af} - \tau_{i-1})2^{k-1}$ for $1 \leq k \leq x$; furthermore, $\lambda_A^{x+1} = \lambda_A$. In the rest of this proof, let $\lambda_k := \lambda_A^k$ for $1 \leq k \leq x+1$. For simplicity, we can assume that $\tau_{i-1} = 0$. By rescaling, let $d_{Af} = 1$.

The following facts follow straightforwardly from the definitions and the assumptions:

$$\lambda_k = 2^{k-1}, \text{for each } 1 \leq k \leq x;$$
$$\lambda_{x+1} = (3/4)d_{Al} \leq 2^x.$$

For convenience, let $\lambda_0 = \tau_{i-1}$. We show how each of the jobs in $A \cap \mathcal{J}''$ can be processed entirely within one of these subzones $[\lambda_{k-1}, \lambda_k)$ when we build the schedule $\overline{OPT}_{\mathcal{I}_2}$. The basic idea is this: treat these subzones as bins with capacity equal to their length, and the jobs $j \in A \cap \mathcal{J}''$ as items whose sizes are their processing time $|OPT_{\mathcal{I}_1}(j)|$ divided by a factor of 4. Each item $j \in A \cap \mathcal{J}''$ can only be put into a bin (subzone) whose interval is entirely contained in j's allowed interval I_{j1}. If this can be done, then we have a new schedule where the jobs in $A \cap \mathcal{J}''$ are feasibly processed, i.e., within their allowed intervals in instance $\mathcal{I}_2$, with processing speeds equal to 4 times their original speeds in $OPT_{\mathcal{I}_1}$.

Let $t_j = |OPT_{\mathcal{I}_1}(j)|$ for all jobs $j \in A \cap \mathcal{J}''$. Moreover, let us divide $A \cap \mathcal{J}''$ into disjoint sets $\mathcal{J}_1 \dot\cup \mathcal{J}_2 \dot\cup \cdots \dot\cup \mathcal{J}_{x+1} \dot\cup \mathcal{J}_{x+2}$ as follows. Job $j \in \mathcal{J}_k$ if its finishing time $e_j(OPT_{\mathcal{I}_1}) \in [\lambda_{k-1}, \lambda_k)$ for $1 \leq k \leq x+1$. In case that $e_j(OPT_{\mathcal{I}_1}) \in [\lambda_{x+1}, d_{Al})$, let $j \in \mathcal{J}_{x+2}$.

We note that if $j \in \mathcal{J}_k$ for $k \geq 2$, then j can be feasibly processed within any of the subzones $[\lambda_{k'-1}, \lambda_{k'})$ for any $1 \leq k' \leq k-1$ in instance $\mathcal{I}_2$, since each such subzone $[\lambda_{k'-1}, \lambda_{k'})$ will be one of the allowed intervals of j in $\mathcal{I}_2$ (this follows from the fact that $e_j(OPT_{\mathcal{I}_1}) \geq \lambda_{k-1}$.) Note also that all jobs in $\mathcal{J}_1$ can be feasibly processed within the subzone $[\lambda_0, \lambda_1)$ in instance $\mathcal{I}_2$, since in $\mathcal{I}_1$, all jobs in $A \cap \mathcal{J}''$ have their deadlines at least as late as $d_{Af} = \lambda_1$. These facts are used in the proof of the following claims.

Claim 1. *Suppose that there exists at least one sublandmark within $[\tau_{i-1}, \lambda_A)$. We can process all jobs of $A \cap \mathcal{J}''$ in $\overline{OPT}_{\mathcal{I}_2}$ at four times their speeds in $OPT_{\mathcal{I}_1}$. Moreover, in $\overline{OPT}_{\mathcal{I}_2}$, the following hold:*

1. *Suppose that $x = 1$. Then all jobs in $A \cap \mathcal{J}'' = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}_3$ are feasibly processed within the subzone $[\lambda_0, \lambda_1)$;*
2. *(a) Suppose that $x \geq 2$. Then all jobs in $\mathcal{J}_1 \cup \mathcal{J}_2$ are feasibly processed within the subzone $[\lambda_0, \lambda_1)$.*
   (b) Suppose that $x \geq 3$. Then all jobs in $\mathcal{J}_k$ are feasibly processed within the subzone $[\lambda_{k-2}, \lambda_{k-1})$ for $3 \leq k \leq x$;

(c) Suppose that $x \geq 2$. Then all jobs in $\mathcal{J}_{x+1} \cup \mathcal{J}_{x+2}$ are feasibly processed within the subzones $[\lambda_{x-1}, \lambda_x)$ and/or $[\lambda_x, \lambda_{x+1})$.

Claim 2. *Suppose that there is no sublandmark in $[\tau_{i-1}, \lambda_A)$. We can feasibly process all jobs in $A \cap \mathcal{J}''$ in $\overline{OPT}_{\mathcal{I}_2}$ in the subzone $[\tau_{i-1}, \lambda_A)$ so that their speeds are four times their speeds in $OPT_{\mathcal{I}_1}$.*

Claims 1 and 2 imply that the jobs in $A \cap \mathcal{J}''$ can be feasibly processed within the subzones partitioning the interval $[\tau_{i-1}, \lambda_A)$ with quadrupled speeds. By a symmetric argument, in $\overline{OPT}_{\mathcal{I}_2}$, the jobs in $B \cap \mathcal{J}''$ can be feasibly processed within the subzones partitioning the interval $[\lambda_B, \tau_i)$ with quadrupled speeds. Finally, in $\overline{OPT}_{\mathcal{I}_2}$, by equation (1), all jobs in $C \cap \mathcal{J}''$ can be feasibly processed using the subzone $[\lambda_A, \lambda_B)$ with quadrupled speeds. Using Proposition 1, we can thus conclude that $E(\overline{OPT}_{\mathcal{I}_2}) \leq 4^{\alpha-1} E(OPT_{\mathcal{I}_1})$. Now the entire lemma follows from the fact that $E(OPT_{\mathcal{I}_2}) \leq E(\overline{OPT}_{\mathcal{I}_2})$. □

Transformation III

In this section, we transform $\mathcal{I}_2$ into an instance $\mathcal{I}_3$ for *unrelated machine scheduling* with the objective of minimizing the L_α-norm.

> Unrelated Machine Scheduling with L_α-norm objective
>
> Given m machines $\mathcal{M}$ and n jobs $\mathcal{J}$, where each job j takes p_{ij} processing time on machine i, the goal is to find an assignment A mapping jobs to machines that minimizes $COST(A) = (\sum_{i \in \mathcal{M}} (\sum_{j:A(j)=i} p_{ij})^\alpha)^{1/\alpha}$.

Recall that in $\mathcal{I}_2$, each job $j \in \mathcal{J}$ has a set of allowed intervals, each of which corresponds to a subzone. Let the collection of all subzones be Z and $|z_i|$ denote the length of a subzone $z_i \in Z$. In $\mathcal{I}_3$, each subzone z_i corresponds to a machine $i \in \mathcal{M}$. For each job $j \in \mathcal{J}$ and each subzone z_i, if j does not have an allowed interval corresponding to z_i, then $p_{ij} = +\infty$. If it has such an interval, then let $p_{ij} = \frac{v_j}{|z_i|^{(\alpha-1)/\alpha}}$.

Note that by convexity of the power function, we can assume that a schedule in $\mathcal{I}_2$ processes a set of jobs within a subzone using a uniform speed. The following lemma shows a one-to-one correspondence between a feasible schedule in $\mathcal{I}_2$ and a feasible assignment in $\mathcal{I}_3$.

Lemma 4. *For each assignment A in $\mathcal{I}_3$, a feasible schedule $\mathcal{S}_{\mathcal{I}_2}(A)$ for instance $\mathcal{I}_2$ can be created as follows. If a set of jobs $\mathcal{J}' \subseteq \mathcal{J}$ is assigned to machine $i \in \mathcal{M}$ in A, $\mathcal{S}_{\mathcal{I}_2}(A)$ processes the jobs in $\mathcal{J}'$ with the uniform speed of $\frac{\sum_{j \in \mathcal{J}'} v_j}{|z_i|}$ in subzone z_i. Conversely, given a feasible schedule $\mathcal{S}_{\mathcal{I}_2}(A)$ for instance $\mathcal{I}_2$, let $A_{\mathcal{I}_2}(j)$ denote the subzone in which job j is processed under $\mathcal{S}_{\mathcal{I}_2}(A)$. We can create an assignment A in instance $\mathcal{I}_3$ so that $A(j) = A_{\mathcal{I}_2}(j)$ for all jobs $j \in \mathcal{J}$.*

In both cases, we have $(COST(A))^\alpha = E(\mathcal{S}_{\mathcal{I}_2}(A))$.

Lemma 5. *Let $OPT_{\mathcal{I}_3}$ be an optimal and A be a feasible solution for $\mathcal{I}_3$, such that $COST(A) \leq 2COST(OPT_{\mathcal{I}_3})$, i.e., A is a 2-approximate solution for $\mathcal{I}_3$. Let $\mathcal{S}_{\mathcal{I}_2}(A)$ be the corresponding schedule of A and $OPT_{\mathcal{I}_2}$ be the corresponding schedule of $OPT_{\mathcal{I}_3}$ in instance $\mathcal{I}_2$. Then $E(\mathcal{S}_{\mathcal{I}_2}(A)) \leq 2^\alpha E(OPT_{\mathcal{I}_2})$.*

We can produce a 2-approximation schedule as stated in Lemma 5 using the algorithm of Azar and Epstein [2]. Combining Lemmas 1, 3, and 5, we prove Theorem 1 for the case of laminar instances.

4 General Instances

For a general instance, our algorithm is similar to what we have done for a laminar instance. Recall that we use the following three transformations for a laminar instance $\mathcal{I}$.

1. Transformation I. Introduce a set of landmarks to transform $\mathcal{I}$ to $\mathcal{I}_1$, where each job's allowed interval is chopped up into a set of allowed intervals.
2. Transformation II. Introduce a set of sublandmarks to transform $\mathcal{I}_1$ to $\mathcal{I}_2$, where each allowed interval of a job is possibly shortened and then further chopped up into a set of allowed intervals.
3. Transformation III. Transform $\mathcal{I}_2$ to $\mathcal{I}_3$, an instance of unrelated machine scheduling.

For a general instance $\mathcal{I}$, Transformations II and III can remain unchanged. But we need to revise Transformation I as follows.

- Transformation I_1. We introduce a sweepline algorithm to define the landmarks.
- Transformation I_2. We shorten a subset of the allowed intervals so that the instance has the same structure as we have had after transformation I when dealing with a laminar instance.

Transformation $\mathbf{I_1}$

We define a set of landmarks using the sweepline algorithm *Sweep* presented in Figure 4. Let $\Psi = \cup_{j \in \mathcal{J}} \{r_j, d_j\}$ be the set of *events*. Note that $\Psi \leq 2|\mathcal{J}|$. The order of these events is based on their numerical values, where ties are broken arbitrarily.

Algorithm *Sweep*, sweeps the time horizon from left to right until it meets the first deadline. It then sets this deadline as a landmark. Every job that was seen up to that point gets "removed" and *Sweep* repeats the same procedure.

Let $\tau_0 = 0 < \tau_1 < \cdots < \tau_k < \tau_{k+1} = d_{\max}$ be the set of landmarks defined by the above sweepline algorithm. The following lemma follows easily by an inductive argument.

Lemma 6. *Let $\tau_i \in T_{land}$ be the set of landmarks returned by the algorithm Sweep, where τ_i's are ordered increasingly. Within each zone $[\tau_{i-1}, \tau_i)$, for $1 \leq i < |T_{land}|$, there exists at least a job $j \in \mathcal{J}$ whose allowed interval I_j is completely contained in such a zone, i.e., $I_j \subseteq [\tau_{i-1}, \tau_i)$.*

We create a new instance $\mathcal{I}_1$ based on $\mathcal{I}$ in the same manner as before. For each job $j \in \mathcal{J}$ with its allowed interval $I_j = [r_j, d_j)$, suppose that

$$\tau_{i-1} \leq r_j < \tau_i < \tau_{i+1} < \cdots < \tau_{i+k} < d_j \leq \tau_{i+k+1}.$$

```
**Algorithm Sweep:**
**Initialization:** Set $\mathcal{J}_c := \emptyset$, and events $\Psi := \cup_{j\in\mathcal{J}}\{r_j, d_j\}$. Assume that the elements $t_i, 1 \le t_i \le |\Psi|$ of $\Psi$ are ordered, and let $T_{\text{land}} := \{0, d_{\max}\}$ be the initial set of landmarks.
**For** $i = 1$ **to** $|\Psi|$ **do**:
    **If** $t_i = r_j$ for some job $j \in \mathcal{J}\backslash\mathcal{J}_c$ **then** $\mathcal{J}_c = \mathcal{J}_c \cup \{j\}$.
    **Else If** $t_i = d_j$ for some $j \in \mathcal{J}_c$ **then**
        add $t_i$ to $T_{\text{land}}$; Set $\mathcal{J}_c := \emptyset$.
```

Fig. 4. A sweepline algorithm to define landmarks

Then as in Transformation I, we replace its allowed interval $I_j = [r_j, d_j)$ with a set of allowed intervals $\bigcup_{s=1}^{k+2} I_{js}$, where

$$I_{j1} = [r_j, \tau_i),\ I_{j(k+2)} = [\tau_{i+k}, d_j),\ \text{and } I_{js} = [\tau_{s+i-2}, \tau_{s+i-1}) \text{ for } 2 \le s \le k+1.$$

Lemma 7. *Let $OPT_{\mathcal{I}}$ and $OPT_{\mathcal{I}_1}$ denote the optimal schedules for instances $\mathcal{I}$ and $\mathcal{I}_1$ respectively. Then $E(OPT_{\mathcal{I}_1}) \le 2^{\alpha-1}E(OPT_{\mathcal{I}})$.*

Transformation I$_2$

In $\mathcal{I}_1$, each job has at most one allowed interval in the zone $[\tau_{i-1}, \tau_i)$. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the set of jobs j that have exactly one allowed interval in $[\tau_{i-1}, \tau_i)$. Assume that $j \in \mathcal{J}'$ has the allowed interval $I_{j1} = [r_{j1}, d_{j1})$. As before, we can divide $\mathcal{J}'$ into three groups A, B and C: (1) $j \in A$, if $r_{j1} = \tau_{i-1}$, $d_{j1} < \tau_i$; (2) $j \in B$ if $\tau_{i-1} < r_{j1}$, $d_{j1} = \tau_i$; and (3) $j \in C$ if $r_{j1} = \tau_{i-1}$, $d_{j1} = \tau_i$.

However, unlike the laminar case, the allowed intervals of jobs in groups A and B can overlap (see Figure 2(b)) in instance $\mathcal{I}_1$. Thus the technique employed in Transformation II breaks down.

To deal with this, we shorten the allowed intervals of jobs in groups A and B so that in the new instance $\mathcal{I}_{1,1}$, the allowed intervals of jobs in group A do not overlap with those of jobs in group B. We argue that the cost of the optimal solution in instance $\mathcal{I}_{1,1}$ does not increase by too much compared to the optimal solution in instance $\mathcal{I}_1$.

We now transform $\mathcal{I}_1$ into $\mathcal{I}_{1,1}$ as follows. For the zone $[\tau_{i-1}, \tau_i)$, let again d_{Al} be the latest deadline in group A and r_{Bf} the earliest release time in group B. Suppose that $j \in \mathcal{J}'$ and its allowed interval is $I_{j1} = [r_{j1}, d_{j1})$.

- If $j \in A$ and $d_{j1} > \tau_{i-1} + \frac{d_{Al}-\tau_{i-1}}{2}$, then replace j's allowed interval with $I'_{j1} = [\tau_{i-1}, \tau_{i-1} + \frac{d_{Al}-\tau_{i-1}}{2})$.
- If $j \in B$ and $r_{j1} < \tau_i - \frac{\tau_i - d_{Bf}}{2}$, then replace j's allowed interval with $I'_{j1} = [\tau_i - \frac{\tau_i - r_{Bf}}{2}, \tau_i)$.

Observe that because Lemma 2 still holds for instances $\mathcal{I}_1$ and $\mathcal{I}_{1,1}$, we can assume that the jobs in group A are processed first, then the jobs in group C, and then jobs in groups B. Jobs in group A (resp. group B) are processed in the order of their increasing deadlines (resp. increasing release times). We show the following lemma.

Lemma 8. *Let $OPT_{\mathcal{I}_1}$ and $OPT_{\mathcal{I}_{1,1}}$ denote the optimal schedules for instances $\mathcal{I}_1$ and $\mathcal{I}_{1,1}$ respectively. Then $E(OPT_{\mathcal{I}_{1,1}}) \leq 2^{\alpha-1}E(OPT_{\mathcal{I}})$.*

Combining Lemmas 7, 8, 3, and 5, we complete the proof of Theorem 1.

References

1. Albers, S.: Energy-efficient algorithms. Comm. of the ACM 53(5), 86–96 (2010)
2. Azar, Y., Epstein, A.: Convex programming for scheduling unrelated parallel machines. In: STOC 2005, pp. 331–337 (2005)
3. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM 54, 1 (2007)
4. Bartal, Y., Leonardi, S., Shallom, G., Sitters, R.: On the Value of Preemption in Scheduling. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 39–48. Springer, Heidelberg (2006)
5. Braun, O., Schmid, G.: Parallel processor scheduling with limited number of preemptions. SIAM J. Computing 32, 671–680 (2003)
6. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.-D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
7. Chen, J.-J., Kuo, T.-W., Lu, H.-I.: Power-Saving Scheduling for Weakly Dynamic Voltage Scaling Devices. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 338–349. Springer, Heidelberg (2005)
8. Chuzhoy, J., Codenotti, P.: Resource Minimization Job Scheduling. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 70–83. Springer, Heidelberg (2009)
9. Chuzhoy, J., Ostrovsky, R., Rabani, Y.: Approximation algorithms for the job interval selection problem and related scheduling problems. Math. Oper. Res. 31(4), 730–738 (2006)
10. Dimpsey, R.T., Iyer, R.K.: Performance degradation due to multiprogramming and system overheads in real workloads: Case study on a shared memory multiprocessor. In: International Conference on Supercomputing 1990, pp. 227–238 (1990)
11. Etsion, Y., Tsafrir, D., Feitelson, D.G.: Effects of clock resolution on the scheduling of interactive and soft real-time processes. In: SIGMETRICS 1990, pp. 172–183 (1990)
12. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-Completeness. W.H. Freeman (1979)
13. Natarajan, C., Sharma, S., Iyer, R.K.: Measurement-based characterization of global memory and network connection, operating system and parallelization overheads: Case study on a shared memory multiprocessor. In: Annual International Symposium on Computer Architecture, vol. 21, pp. 71–80 (1994)
14. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. J. Combinatorial Optimization 11, 305–319 (2006)
15. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM J. on Computing 35, 658–671 (2005)
16. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS 1995, 374-382 (1995)

A Fast Algorithm for Permutation Pattern Matching Based on Alternating Runs

Marie-Louise Bruner[1] and Martin Lackner[2,⋆]

[1] Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Austria
marie-louise.bruner@tuwien.ac.at
[2] Institute of Information Systems, Vienna University of Technology, Austria
lackner@dbai.tuwien.ac.at

Abstract. The NP-complete PERMUTATION PATTERN MATCHING problem asks whether a permutation P can be matched into a permutation T. A matching is an order-preserving embedding of P into T. We present a fixed-parameter algorithm solving this problem with an exponential worst-case runtime of $\mathcal{O}^*(1.79^{\mathsf{run}(T)})$, where $\mathsf{run}(T)$ denotes the number of alternating runs of T. This is the first algorithm that improves upon the $\mathcal{O}^*(2^n)$ runtime required by brute-force search without imposing restrictions on P and T. Furthermore we prove that – under standard complexity theoretic assumptions – such a fixed-parameter tractability result is not possible for $\mathsf{run}(P)$.

1 Introduction

The concept of pattern avoidance (and, closely related, pattern matching) in permutations arose in the late 1960ies. It was in an exercise of his *Fundamental algorithms* [12] that Knuth asked which permutations could be sorted using a single stack. The answer is simple: These are exactly the permutations avoiding the pattern 231 and they are counted by the Catalan numbers. By avoiding (resp. containing) a certain pattern the following is meant: The permutation $\pi = 53142$ (written in one-line representation) contains the pattern 231, since the subsequence 342 of π is order-isomorphic to 231. We call the subsequence 342 a matching of 231 into π. On the other hand, π avoids the pattern 123 since it contains no increasing subsequence of length three. Since 1985, when the first systematic study of *Restricted Permutations* [17] was published by Simion and Schmidt, the area of pattern avoidance in permutations has become a rapidly growing field of discrete mathematics, more specifically of (enumerative) combinatorics [4, 11].

This paper takes the viewpoint of computational complexity. Computational aspects of pattern avoidance, in particular the analysis of the PERMUTATION PATTERN MATCHING (PPM) problem, have received far less attention than enumerative questions until now. The PPM problem is defined as follows:

PERMUTATION PATTERN MATCHING (PPM)
Instance: A permutation T (the text) of length n and a permutation P (the pattern) of length $k \leq n$.
Question: Is there a matching of P into T?

⋆ The second author was supported by the Austrian Science Fund (FWF): P20704-N18.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 261–270, 2012.

In [5] it was shown that PPM is in general NP-complete. From this result follows a trivial brute-force algorithm checking every length k subsequence of T. Its runtime is in $\mathcal{O}^*(2^n)$, i.e. is bounded by $2^n \cdot poly(n)$. To the best of our knowledge, no algorithm with a runtime of $\mathcal{O}^*((2-\epsilon)^n)$ without restrictions on P and T is known yet. If such restrictions are imposed, improvements have been achieved. There are polynomial time algorithms in case of a *separable* pattern [2,5,10]. Separable permutations avoid both 3142 and 2413. In case P is the identity $12\dots k$, PPM consists of looking for an increasing subsequence of length k in the text – this is a special case of the LONGEST INCREASING SUBSEQUENCE problem. This problem can be solved in $\mathcal{O}(n \log n)$-time for sequences in general [16] and in $\mathcal{O}(n \log\log n)$-time for permutations [7,14]. PPM can be solved in $\mathcal{O}(n \log n)$-time for all patterns of length four [2]. An $\mathcal{O}(k^2 n^6)$-time algorithm is presented in [9] for the case that both the text and the pattern are 321-avoiding.

In this paper we tackle the problem of solving PPM faster than $\mathcal{O}^*(2^n)$ for arbitrary P and T. We achieve this by exploiting the decomposition of permutations into alternating runs. As an example, the permutation $\pi = 53142$ has three alternating runs: 531 (down), 4 (up) and 2 (down). We denote this number of ups and downs in a permutation π by $\mathsf{run}(\pi)$. Alternating runs are a fundamental permutation statistic and had been studied already in the late 19th century by André [3]. An important result was the characterization of the distribution of $\mathsf{run}(\pi)$ in a random permutation: asymptotically, $\mathsf{run}(\pi)$ is normal with mean $\frac{1}{3}(2|\pi|-1)$ [13]. Despite the importance of alternating runs within the study of permutations, the connection to PPM has so far not been explored.

In detail the contributions of this paper are the following:

- We present a fixed-parameter algorithm for PPM with an exponential runtime of $\mathcal{O}^*(1.79^{\mathsf{run}(T)})$. Since the combinatorial explosion is confined to $\mathsf{run}(T)$, this algorithm performs especially well when T has few alternating runs. Indeed, the runtime depends only polynomially on n, the length of T.
- Since $\mathsf{run}(T) \leq n$, this algorithm also solves PPM in time $\mathcal{O}^*(1.79^n)$. This is a major improvement over the brute-force algorithm.
- Furthermore, we analyze this algorithm with respect to $\mathsf{run}(P)$. We obtain a runtime of $\mathcal{O}^*\left(\left(n^2/2\mathsf{run}(P)\right)^{\mathsf{run}(P)}\right)$. In the framework of parameterized complexity theory this runtime proves XP membership for $\mathsf{run}(P)$.[1]
- Finally, we prove that this XP result cannot be substantially improved. We prove that – under standard complexity theoretic assumptions – no fixed-parameter algorithm exists with respect to $\mathsf{run}(P)$, i.e. no algorithm with runtime $\mathcal{O}^*(c^{\mathsf{run}(P)})$ for some constant c may be hoped for.

Proofs had to be omitted in this version – we refer the reader to the full version of this paper [6]. Runtime calculations can be found there as well.

2 Preliminaries

Permutations. For any $m \in \mathbb{N}$, let $[m]$ denote the set $\{1,\dots,m\}$ and $[0,m]$ denote $\{0,1,\dots,m\}$. A permutation π on the set $[m]$ can be seen as the sequence $\pi(1), \pi(2)$,

[1] XP membership also follows from results in [1] and a lemma shown in [6].

$\ldots, \pi(m)$. Viewing permutations as sequences allows us to speak of *subsequences* of a permutation. We speak of a *contiguous subsequence* of π if the sequence consists of contiguous elements in π.

Definition 1. *Let P (the pattern) be a permutation of length k. We say that the permutation T (the text) of length n* contains P as a pattern *or that P* can be matched into T *if we can find a subsequence of T that is order-isomorphic to P. If there is no such subsequence we say that T* avoids the pattern P. *Matching P into T thus consists in finding a monotonically increasing map $\varphi : [k] \to [n]$ so that the sequence $\varphi(P)$, defined as $\big(\varphi(P(i))\big)_{i\in[k]}$, is a subsequence of T.*

Every permutation π on $[m]$ defines a total order $\prec_\pi$ on $[m]$. We write $i \prec_\pi j$ iff $\pi^{-1}(i) < \pi^{-1}(j)$, i.e. the value i stands to the left of the value j in π. When considering the minimum (maximum) of a subset $\mathcal{S} \subseteq [m]$ with respect to $\prec_\pi$, we write $\min_\pi \mathcal{S}$ ($\max_\pi \mathcal{S}$).

We discern two types of local extrema in permutations: valleys and peaks. A *valley* of a permutation π is an element $\pi(i)$ for which it holds that $\pi(i-1) > \pi(i)$ and $\pi(i) < \pi(i+1)$. If $\pi(i-1)$ or $\pi(i+1)$ is not defined, we still speak of valleys. The set $\mathit{Val}(\pi)$ contains all valleys of π. Similarly, a *peak* denotes an element $\pi(i)$ for which it holds that $\pi(i-1) < \pi(i)$ and $\pi(i) > \pi(i+1)$.

Valleys and peaks partition a permutation into contiguous monotone subsequences, so-called *(alternating) runs*. The first run of a given permutation starts with its first element (which is also the first local extremum) and ends with the second local extremum. The second run starts with the following element and ends with the third local extremum. Continuing in this way, every element of the permutation belongs to exactly one alternating run. Observe that every alternating run is either increasing or decreasing. We therefore distinguish between *runs up* and *runs down*. Note that runs up always end with peaks and runs down always end with valleys. The parameter $\mathsf{run}(\pi)$ counts the number of alternating runs in π. Hence $\mathsf{run}(\pi)+1$ equals the number of local extrema in π. These definitions can be analogously extended to subsequences of permutations.

Example 2. In the permutation $1\,8\,12\,4\,7\,11\,6\,3\,2\,9\,5\,10$ the valleys are 1, 4, 2 and 5 and the peaks are 12, 11, 9 and 10. A decomposition into alternating runs is given by: $1\,8\,12|4|7\,11|6\,3\,2|9|5|10$. A graphical representation can be found in Figure 1 on page 265. $\dashv$

Parameterized Complexity Theory. In contrast to classical complexity theory, a parameterized complexity analysis studies the runtime of an algorithm with respect to an additional parameter and not just the input size $|I|$. A problem parameterized by a parameter p is *fixed-parameter tractable* (or in FPT) if there is an algorithm solving it in time $\mathcal{O}(f(p)\cdot|I|^c)$, where f is a computable function and c a constant. The algorithm itself is also called fixed-parameter tractable (fpt). In this paper we want to focus on the exponential runtime of algorithms, i.e. the function f, and therefore use the $\mathcal{O}^*$ notation which neglects polynomial factors. The classes $\mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \ldots$ build the so-called W-hierarchy. It is conjectured (and widely believed) that $\mathsf{W[1]} \neq \mathsf{FPT}$. Therefore showing $\mathsf{W[1]}$-hardness can be considered as evidence that a problem is not fixed-parameter tractable. A problem is in XP with respect to a parameter k if it can be solved in time $\mathcal{O}(|I|^{f(k)})$ where f is a computable function. It holds that $\mathsf{FPT} \subseteq \mathsf{W[1]} \subseteq \mathsf{W[2]} \subseteq \ldots \subseteq \mathsf{XP}$. For details we refer the reader to [8,15].

3 The Alternating Run Algorithm

We start with an outline of the alternating run algorithm. Its description consists of two parts. In Part 1 we introduce so-called *matching functions*. These functions map runs in P to sequences of adjacent runs in T. The intention behind matching functions is to restrict the search space to certain length k subsequences, namely to those where all elements in a run in P are mapped to elements in the corresponding sequences of runs in T. In Part 2 a dynamic programming algorithm is described. It checks for every matching function whether it is possible to find a compatible matching. This is done by finding a small set of representative elements to which the element 1 can be mapped to, then – for a given choice for 1 – finding representative values for 2, and so on.

Theorem 3. *The alternating run algorithm solves* PPM *in time* $\mathcal{O}^*(1.79^{\mathsf{run}(T)})$. *Therefore* PPM *parameterized by* $\mathsf{run}(T)$ *is in* FPT.

Corollary 4. *The alternating run algorithm solves* PPM *in time* $\mathcal{O}^*(1.79^n)$ *where n is the length of the text T.*

Proposition 5. PPM *is in* XP *with respect to the parameter* $\mathsf{run}(P)$ *since the alternating run algorithm solves* PPM *in time* $\mathcal{O}^*\left(\left(\frac{n^2}{2\mathsf{run}(P)}\right)^{\mathsf{run}(P)}\right)$.

Throughout this section the input instance (T_{ex}, P_{ex}) which is given by $T_{ex} = 1\,8\,12\,4\,7\,11\,6\,3\,2\,9\,5\,10$ and $P_{ex} = 2\,3\,1\,4$ serves as a running example.

Part 1: Matching Functions. We introduce the concept of matching functions. These are functions from $[\mathsf{run}(P)]$, i.e. runs in P, to sequences of adjacent runs in T. For a given matching function F the search space in T is restricted to matchings where an element i contained in the j-th run in P is matched to an element in $F(j)$. Two adjacent runs in P are mapped to sequences of runs that overlap with exactly one run. This overlap is necessary since elements in different runs in P may be matched to elements in the same run in T. More precisely, valleys and peaks in P might be matched to the same run in T as their successors (see the following example).

Example 6. In Figure 1 P_{ex} (left-hand side) and T_{ex} (right-hand side) are depicted together with a matching function F. A matching compatible with F is given by $4\,6\,2\,9$. We can see that the elements 6 and 2 lie in the same run in T_{ex} even though 3 (a peak) and 1 (its successor) lie in different runs in P_{ex}. ⊣

Definition 7. *A matching function F maps an element of* $[\mathsf{run}(P)]$ *to a subsequence of T. It has to satisfy the following properties for all $i \in$* $[\mathsf{run}(P)]$.

(P1) $F(i)$ is a contiguous subsequence of T.

(P2) If the i-th run in P is a run up (down), $F(i)$ starts with an element following a valley (peak) or the first element in T and ends with a valley (peak) or the last element in T.

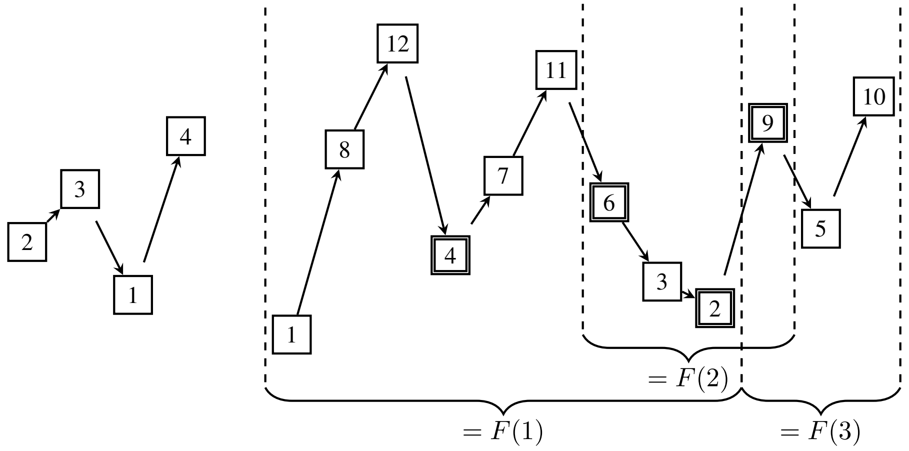


Fig. 1. P_{ex} and T_{ex} together with a matching function F and the compatible matching 4 6 2 9

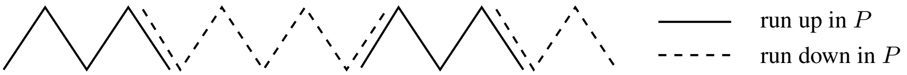


Fig. 2. A sketch of a matching function and its M- and W-shaped subsequences

(P3) $F(1)$ *starts with the first and* $F(\mathsf{run}(P))$ *ends with the last element in* T.

(P4) $F(i)$ *and* $F(i+1)$ *have one run in common:* $F(i+1)$ *starts with the leftmost element in the last run in* $F(i)$.

Property (P2) implies that every run up is matched into an M-shaped sequence of runs of the form up-down-up-...-up-down (if the run up is the first or the last run in P the sequence might start or end differently) and every run down is matched into a W-shaped sequence of runs of the form down-up-down-...-down-up (again, if the run down is the first or the last run in P, the sequence might start or end differently). These M- and W-shaped sequences and their overlap are sketched in Figure 2.

The following lemma is essential as it enables us to iterate over all matching functions in fpt time.

Lemma 8. *There are at most* $\sqrt{2}^{\mathsf{run}(T)}$ *functions from* $[\mathsf{run}(P)]$ *to subsequences of* T *that satisfy (P1) to (P4).*

Proof idea. A matching function F can be uniquely determined by fixing the first run up in each $F(i)$. There are at most $\lceil \mathsf{run}(T)/2 \rceil$ runs up in T. □

Part 2: Finding a Matching. When checking whether T contains P as a pattern, it is sufficient to test for all matching functions whether there exists a *compatible* matching. A matching is compatible with a matching function F if an element i contained in the j-th run in P is matched to an element in $F(j)$. This is checked by a dynamic programming algorithm. The algorithm computes the data structure X_κ for each $\kappa \in$

$[k]$. X_κ is a subset of $[0,n]^{\mathsf{run}(P)}$ and contains representative choices for the matching of the largest element in each run in P that is $\leq \kappa$. (X_κ does of course depend on F but we omit this in the notation.)

Let us explain what is meant by representative choices. We search for a compatible matching of P into T by successively determining possible elements for $1, 2, \ldots, k$. Given a choice for $\kappa \in [k]$, possible choices for $\kappa + 1$ are necessarily larger. In addition, it is always preferable to choose elements that are as small as possible. To be more precise: if $\nu \in [n]$ has been chosen for $\kappa \in [k]$, we merely need to consider the valleys of the subsequence of T containing all elements larger than ν. Indeed, if any matching of P into T can be found, it is also possible to find a matching that only involves valleys in the above-mentioned subsequences. Therefore our algorithm will only consider such valleys – we call these elements representative. As an example, consider again Figure 1. Here $4\,6\,3\,10$ is a matching of P_{ex} into T_{ex} where the elements 3 and 10 are not representative. This can be seen since 3 is not a valley and 10 is not a valley in the subsequence consisting of elements larger than 6. However, this matching can be represented by the matching $4\,6\,2\,9$ that only involves representative elements (3 is represented by 2; 10 by 9).

Furthermore, observe that when successively determining possible elements for $1, 2, \ldots, k$, we move from left to right in runs up and from right to left in runs down. Hence the chosen elements do not only have to be larger than the previously chosen element but also have to lie on the correct side of the previously chosen element in the same run. These observations are captured in the following definition.

Definition 9. *For a permutation π on $[n]$ and integers $i, j \leq n$, we define $\pi_{U(i,j)}$ ($\pi_{D(i,j)}$) as the subsequence of π consisting of all elements that are right (left) of j and larger than i. Then $URep(\pi,i,j) := Val(\pi_{U(i,j)})$ (resp. $DRep(\pi,i,j) := Val(\pi_{D(i,j)})$) corresponds to the set of* representative elements *for the case of a run up (resp. down).*

For an example, see Figure 3 where representative elements are shown for the permutation T_{ex}, $i = 3$ and $j = 2$.

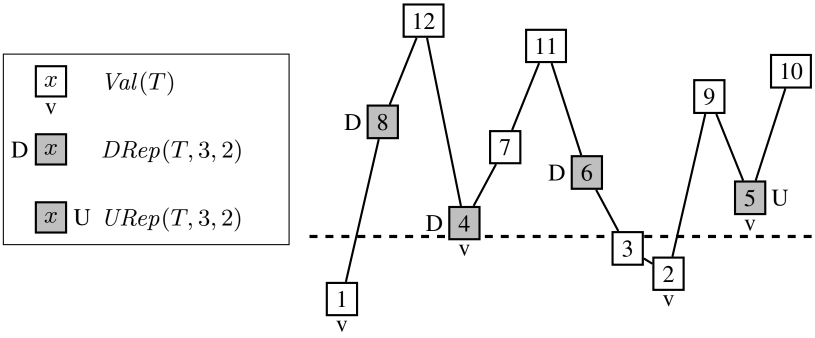


Fig. 3. Illustrating Definition 9

We now describe how the algorithm checks whether there is a matching from P into T compatible with the matching function F. The data structure X_κ consists of $\mathsf{run}(P)$-tuples with entries in $[0,n]$. The i-th component of a tuple in X_κ is a representative choice for the largest element $\leq \kappa$ that lies in the i-th run. Limiting X_κ to representative elements allows to bound its size by $1.262^{\mathsf{run}(T)}$. In order to achieve this upper bound, we impose the following conditions (C1) and (C2) and remove unnecessary elements by applying the rules (R1) and (R2). In order to state these conditions and rules, we write $r(\kappa) = i$ iff κ is contained in the i-th run in P. For notational convenience we define $r(0) := 1$.

First, we set $X_0 := \{(0,0,\ldots,0)\}$. The set X_κ is then constructed from $X_{\kappa-1}$ as follows. Let $\mathbf{x} = (x_1,\ldots,x_{\mathsf{run}(P)}) \in X_{\kappa-1}$. We now define $N_{\kappa,\mathbf{x}}$ to be the set of all $\nu \in [n]$ that satisfy (C1) and (C2). This set contains representative elements to which κ may be mapped to for the given $\mathbf{x}$.

(C1) It has to hold that $\nu \in \mathit{URep}(F(r(\kappa)), x_{r(\kappa-1)}, x_{r(\kappa)})$ in case κ lies in a run up and analogously $\nu \in \mathit{DRep}(F(r(\kappa)), x_{r(\kappa-1)}, x_{r(\kappa)})$ in case κ lies in a run down.

This condition ensures that ν is larger than the previously chosen element for $\kappa - 1$, i.e. larger than $x_{r(\kappa-1)}$. Furthermore, it enforces ν to lie on the correct side of $x_{r(\kappa)}$, the previously chosen element in this run. Instead of considering all such elements in $F(r(\kappa))$ we only take into account representative elements.

(C2) If κ is *not* the largest element in its run in P, there has to exist $\xi \in F(r(\kappa))$ with $\nu < \xi$ and $\nu \prec_T \xi$ for κ appearing in a run up ($\xi \prec_T \nu$ for κ appearing in a run down).

This condition excludes a choice for κ that cannot lead to a matching. A non-maximal element in a run up (down) in P has to be mapped to an element having larger elements to its right (left). We therefore exclude elements in the rightmost (leftmost) run of $F(r(\kappa))$ if this is a run down (up). Condition (C2) is necessary to obtain the runtime bounds for the dynamic programming algorithm.

As an intermediate step let $X'_\kappa := \{\mathbf{x}(\nu) \mid \mathbf{x} \in X_{\kappa-1} \text{ and } \nu \in N_{\kappa,\mathbf{x}}\}$, where $\mathbf{x}(\nu) := (x_1,\ldots,x_{r(\kappa)-1},\nu,x_{r(\kappa)+1},\ldots,x_{\mathsf{run}(P)})$. The tuple $\mathbf{x}(\nu)$ thus differs from $\mathbf{x}$ only at the $r(\kappa)$-th position. Note that two different elements $\mathbf{x}$ and $\mathbf{x}'$ in $X_{\kappa-1}$ may lead to the same element $\mathbf{x}(\nu) = \mathbf{x}'(\nu)$ in X_κ if they only differ in $x_{r(\kappa)}$. Rule (R1) describes how to compute X_κ from X'_κ. Stating it requires the following definition.

Definition 10. *Let π be a permutation of length n. A subsequence of π consisting of a consecutive run down and run up (formed like a V) is called a* vale. *If π starts with a run up, this run is also considered as a vale and analogously if π ends with a run down. For two elements $\nu_1, \nu_2 \in [n]$, $\nu_1 \sim \nu_2$ if both lie in the same vale*[2]. *For two k-tuples $\mathbf{x}, \mathbf{y} \in [n]^k$, $\mathbf{x} \sim \mathbf{y}$ if for every $i \in [k]$ it holds that $x_i \sim y_i$. For a fixed set of k-tuples S and $\mathbf{x} \in S$, the equivalence class $[\mathbf{x}]_\sim$ is defined as all $\mathbf{y} \in S$ with $\mathbf{x} \sim \mathbf{y}$.*

(R1) We set $X_\kappa := \{\min_{(r(\kappa))}([\mathbf{x}]_\sim) \mid \mathbf{x} \in X'_\kappa\}$, where $\min_{(i)}(S)$ is the function picking the tuple in S with the smallest value at the i-th position. If this minimum is not unique, it arbitrarily picks one candidate.

[2] Note that every element in a permutation is contained in exactly one vale.

This rule is the key to prove the $1.262^{\mathsf{run}(T)}$ upper bound on $|X_\kappa|$. It is based on the observation that it is enough to keep a single tuple for each $[\mathbf{x}]_\sim$. This means that for a set of tuples with coinciding vales it is enough to consider one of them. We provide an intuition about the rule and its correctness in the following example.

Example 11. Consider the text permutation schematically represented in Figure 4. We are searching for representative choices for κ, an element lying in a run down. For κ', the previous element lying in the same run as κ, two representative elements are μ_1 (circle) and μ_2 (square). They lead to one representative element for $\kappa - 1$ each: if μ_1 has been chosen ν_1 is a representative element (circle) and if μ_2 has been chosen ν_2 is one. Following condition (C1), we find three representative elements for κ in $F(r(\kappa))$: ξ_1 (if ν_1 has been chosen), ξ_2 and ξ_3 (if ν_2 has been chosen).

We can now observe that it is not necessary to store all three representative elements for κ. Indeed, in the vale containing ξ_1 and ξ_2 we only need to keep track of ξ_1 since this is always a better choice than ξ_2. This can be seen in the following way: In general, elements that lie further to the right (left) in a run down (up) might be preferable since they leave more possibilities for future elements that are to be matched. Within a vale however, the horizontal position does not make any difference, it is only the vertical position that matters. Here, the elements left of ξ_2 and right of ξ_1 are not available for following choices even if we choose ξ_2 since they are smaller than ξ_2. However, the elements left of ξ_1 that are smaller than ξ_2 are only available if we choose ξ_1. $\dashv$

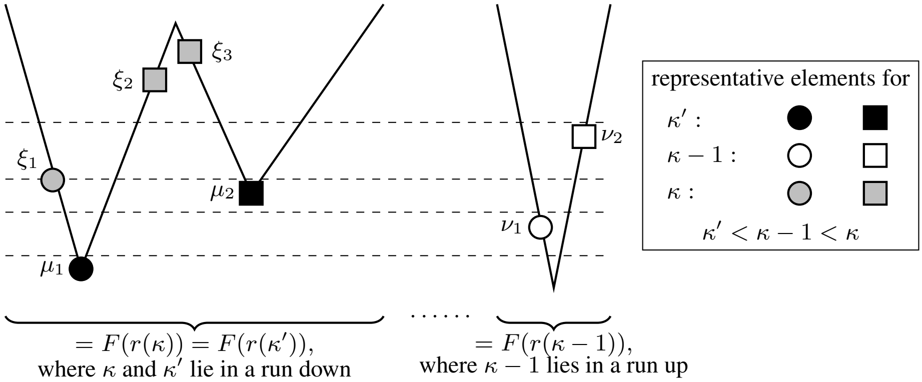

ξ_3
ξ_2
ξ_1
μ_2
μ_1
ν_2
ν_1
representative elements for
κ' : ● ■
$\kappa - 1$: ○ □
κ : ◍ ▨
$\kappa' < \kappa - 1 < \kappa$
$= F(r(\kappa)) = F(r(\kappa'))$,
where κ and κ' lie in a run down
......
$= F(r(\kappa - 1))$,
where $\kappa - 1$ lies in a run up

Fig. 4. Illustrating Rule (R1)

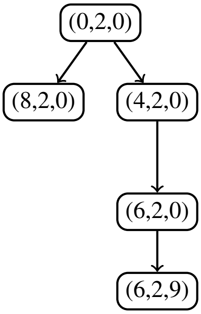
In the case that κ is the largest element in its run, it is enough to consider a single representative element in $F(r(\kappa))$. This is because the position of the element ν is no longer relevant since no further elements have to be chosen in this run. Hence the following data reduction is performed on X_κ.

(R2) Let $M_{\kappa,\mathbf{x}} := \big\{y_{r(\kappa)} \mid \mathbf{y} \in X_\kappa \wedge (y_i = x_i \ \forall i \neq r(\kappa))\big\}$. If κ is the largest element in its run, each $\mathbf{x} = \big(x_1, \ldots, x_{\mathsf{run}(P)}\big) \in X_\kappa$ is replaced by the tuple $\big(x_1, \ldots, x_{r(\kappa)-1}, \min(M_{\kappa,\mathbf{x}}), x_{r(\kappa)+1}, \ldots, x_{\mathsf{run}(P)}\big)$.

As a consequence there are no two tuples in X_κ that only differ at the $r(\kappa)$-th position in this case.

Termination. For a given matching function F, the algorithm described in Part 2 terminates as soon as we have reached X_k. Observe that X_k is always empty if a previous X_κ was empty. If for any F the data structure X_k is non-empty, P can be matched into T.

Example 12. Let us demonstrate with the help of a simple example how the alternating run algorithm works. Consider the text T_{ex} and the pattern P_{ex}. In this example we consider the matching function F represented in Figure 1. Figure 5 depicts a successful run of the algorithm finding the matching $4\,6\,2\,9$. $\dashv$



The only valley in $F(r(1)) = F(2)$ is 2, therefore $N_{1,(0,0,0)} = \{2\}$.

There are 3 representative elements larger than 2 in $F(1)$: 8, 4 and 3. Since 2 is not the largest element in its run in P_{ex}, condition (C2) implies that 3 is ruled out. Thus $N_{2,(0,2,0)} = \{8,4\}$. Rule (R1) yields $X_2 = X_2'$. Rule (R2) is not applicable.

We have $N_{3,(8,2,0)} = \{11\}$ and $N_{3,(4,2,0)} = \{7,6\}$ implying $X_3' = \{(11,2,0),(7,2,0),(6,2,0)\}$. Rule (R1) discards $(11,2,0)$ in favor of $(7,2,0)$. Finally, Rule (R2) is applicable here and discards $(7,2,0)$.

The only representative element larger than 6 in $F(3)$ is 9. The matching of $P_{ex} = 2314$ into T_{ex} found by the algorithm is thus 4629.

Fig. 5. The construction of $X_1, \ldots, X_4$ for our running example (T_{ex}, P_{ex})

4 W[1]-Hardness for the Parameter run(P)

Proposition 5 shows that the alternating run algorithm also yields an XP result with respect to $\mathsf{run}(P)$. The following theorem implies that this result cannot be improved to an FPT result – unless FPT $=$ W[1]. This is shown by an fpt-reduction from the W[1]-complete CLIQUE problem.

Theorem 13. PPM *is* W[1]*-hard with respect to the parameter* $\mathsf{run}(P)$.

5 Future Work

Theorem 3 shows fixed-parameter tractability of PPM with respect to $\mathsf{run}(T)$. An immediate consequence is that any PPM instance can be reduced by polynomial time preprocessing to an equivalent instance – a kernel – of size depending solely on $\mathsf{run}(T)$. This raises the question whether even a polynomial-sized kernel exists. Another research direction is the study of further permutation statistics. The major open problem in this regard is whether PPM is fpt with respect to the length of P. Finally, our method of making use of alternating runs might lead to fast algorithms for other permutation based problems as well.

References

1. Ahal, S., Rabinovich, Y.: On complexity of the subpattern problem. SIAM J. Discrete Math. 22(2), 629–649 (2008)
2. Albert, M., Aldred, R., Atkinson, M., Holton, D.: Algorithms for Pattern Involvement in Permutations. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 355–367. Springer, Heidelberg (2001)
3. André, D.: Étude sur les maxima, minima et séquences des permutations. Ann. Sci. École Norm. Sup. 3(1), 121–135 (1884)
4. Bona, M.: Combinatorics of permutations. Discrete Mathematics and Its Applications. Chapman & Hall/CRC (2004)
5. Bose, P., Buss, J.F., Lubiw, A.: Pattern matching for permutations. Information Processing Letters 65(5), 277–283 (1998)
6. Bruner, M.L., Lackner, M.: A fast algorithm for permutation pattern matching based on alternating runs. CoRR (2012)
7. Chang, M.S., Wang, F.H.: Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs. Information Processing Letters 43(6), 293–295 (1992)
8. Flum, J., Grohe, M.: Parameterized complexity theory. Springer, Heidelberg (2006)
9. Guillemot, S., Vialette, S.: Pattern Matching for 321-Avoiding Permutations. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 1064–1073. Springer, Heidelberg (2009)
10. Ibarra, L.: Finding pattern matchings for permutations. Information Processing Letters 61(6), 293–295 (1997)
11. Kitaev, S.: Patterns in Permutations and Words. Springer, Heidelberg (2011)
12. Knuth, D.E.: The Art of Computer Programming. Fundamental Algorithms, vol. I. Addison-Wesley (1968)
13. Levene, H., Wolfowitz, J.: The covariance matrix of runs up and down. The Annals of Mathematical Statistics 15(1), 58–69 (1944)
14. Mäkinen, E.: On the longest upsequence problem for permutations. International Journal of Computer Mathematics 77(1), 45–53 (2001)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics And Its Applications. Oxford University Press (2006)
16. Schensted, C.: Longest increasing and decreasing subsequences. Classic Papers in Combinatorics, pp. 299–311 (1987)
17. Simion, R., Schmidt, F.W.: Restricted permutations. European Journal of Combinatorics 6, 383–406 (1985)

Sorted Range Reporting*

Yakov Nekrich and Gonzalo Navarro

Department of Computer Science, University of Chile
yakov.nekrich@googlemail.com, gnavarro@dcc.uchile.cl

Abstract. We consider a variant of the orthogonal range reporting problem when all points should be reported in the sorted order of their x-coordinates. We show that reporting two-dimensional points with this additional condition can be organized (almost) as efficiently as the standard range reporting. Moreover, our results generalize and improve the previously known results for the orthogonal range successor problem and can be used to obtain better solutions for some stringology problems.

1 Introduction

An orthogonal range reporting query Q on a set of d-dimensional points S asks for all points $p \in S$ that belong to the query rectangle $Q = \prod_{i=1}^{d}[a_i, b_i]$. The orthogonal range reporting problem, that is, the problem of constructing a data structure that supports such queries, was studied extensively; see for example [1]. In this paper we consider a variant of the two-dimensional range reporting in which reported points must be sorted by one of their coordinates. Moreover, our data structures can also work in the online modus: the query answering procedure reports all points from $S \cap Q$ in increasing x-coordinate order until the procedure is terminated or all points in $S \cap Q$ are output.[1]

Some simple database queries can be represented as orthogonal range reporting queries. For instance, identifying all company employees who are between 20 and 40 years old and whose salary is in the range $[r_1, r_2]$ is equivalent to answering a range reporting query $Q = [r_1, r_2] \times [20, 40]$ on a set of points with coordinates (salary, age). Then reporting employees with the salary-age range Q sorted by their salary is equivalent to a sorted range reporting query.

Furthermore, the sorted reporting problem is a generalization of the orthogonal range successor problem (also known as the range next-value problem) [15,8,14,7,21]. The answer to an orthogonal range successor query $Q = [a, +\infty] \times [c, d]$ is the point with smallest x-coordinate[2] among all points that are in the rectangle Q. The best previously known $O(n)$ space data structure for the range successor queries uses $O(n)$ space and supports queries in

* Partially funded by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F, Chile.

[1] We can get increasing/decreasing x/y-coordinate ordering via coordinate changes.

[2] Previous works (e.g., [8,21]) use slightly different definitions, but all of them are equivalent up to a simple change of coordinate system or reduction to rank space [11].

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 271–282, 2012.

$O(\log n/\log\log n)$ time [21]. The fastest previously described structure supports range successor queries in $O(\log\log n)$ time but needs $O(n\log n)$ space. In this paper we show that these results can be significantly improved.

In Section 3 we describe two data structures for range successor queries. The first structure needs $O(n)$ space and answers queries in $O(\log^{\varepsilon} n)$ time; henceforth ε denotes an arbitrarily small positive constant. The second structure needs $O(n\log\log n)$ space and supports queries in $O((\log\log n)^2)$ time. Both data structures can be used to answer sorted reporting queries in $O((k+1)\log^{\varepsilon} n)$ and $O((k+1)(\log\log n)^2)$ time, respectively, where k is the number of reported points. In Sections 4 and 5 we further improve the query time and describe a data structure that uses $O(n\log^{\varepsilon} n)$ space and supports sorted reporting queries in $O(\log\log n + k)$ time. As follows from the reduction of [17] and the lower bound of [19], any data structure that uses $O(n\log^{O(1)} n)$ space needs $\Omega(\log\log n+k)$ time to answer (unsorted) orthogonal range reporting queries. Thus we achieve optimal query time for the sorted range reporting problem. We observe that the currently best data structure for unsorted range reporting in optimal time [5] also uses $O(n\log^{\varepsilon} n)$ space. In Section 6 we discuss applications of sorted reporting queries to some problems related to text indexing and some geometric problems.

Our results are valid in the word RAM model. Unless specified otherwise, we measure the space usage in words of $\log n$ bits. We denote by $p.x$ and $p.y$ the coordinates of a point p. We assume that points lie on an $n\times n$ grid, i.e., that point coordinates are integers integers in $[1,n]$. We can reduce the more general case to this one by reduction to rank space [11]. The space usage will not change and the query time would increase by an additive factor $pred(n)$, where $pred(n)$ is the time needed to search in a one-dimensional set of integers [20,19].

2 Compact Range Trees

The range tree is a handbook data structure frequently used for various orthogonal range reporting problems. Its leaves contain the x-coordinates of points; a set $S(v)$ associated with each node v contains all points whose x-coordinates are stored in the subtree rooted at v. We will assume that points of $S(v)$ are sorted by their y-coordinates. $S(v)[i]$ will denote the i-th point in $S(v)$; $S(v)[i..j]$ will denote the sorted list of points $S(v)[i], S(v)[i+1], \ldots, S(v)[j]$.

A standard range tree uses $O(n\log n)$ space, but this can be reduced by storing compact representations of sets $S(v)$. We will need to support the following two operations on compact range trees. Given a range $[c,d]$ and a node v, $noderange(c,d,v)$ finds the range $[c_v,d_v]$ such that $p.y\in[c,d]$ if and only if $p\in S(v)[c_v..d_v]$ for any $p\in S(v)$. Given an index i and a node v, $point(v,i)$ returns the coordinates of point $S(v)[i]$.

Lemma 1. *[6,5] There exists a compact range tree that uses $O(nf(n))$ space and supports operations $point(v,i)$ and $noderange(c,d,v)$ in $O(g(n))$ and $O(g(n)+\log\log n)$ time, respectively, for (i) $f(n)=O(1)$ and $g(n)=O(\log^{\varepsilon} n)$; (ii) $f(n)=O(\log\log n)$ and $g(n)=O(\log\log n)$; (iii) $f(n)=O(\log^{\varepsilon} n)$ and $g(n)=O(1)$.*

Proof: We can support $point(v, i)$ in $O(g(n))$ time using $O(nf(n))$ space as in variants (i) and (iii) using a result from Chazelle [6]; we can support $point(v, i)$ in $O(\log\log n)$ time and $O(n \log\log n)$ space using a result from Chan et al. [5]. In the same paper [5, Lemma 2.4], the authors also showed how to support $noderange(c, d, i)$ in $O(g(n) + \log\log n)$ time and $O(n)$ additional space using a data structure that supports $point(v, i)$ in $O(g(n))$ time. □

3 Sorted Reporting in Linear Space

In this section we show how a range successor query $Q = [a, +\infty] \times [c, d]$ can be answered efficiently. We combine the recursive approach of the van Emde Boas structure [20] with compact structures for range maxima queries. A combination of succinct range minima structures and range trees was also used in [5]. A novel idea that distinguishes our data structure from the range reporting structure in [5], as well as from the previous range successor structures, is binary search on tree levels originally designed for one-dimensional searching [20]. We essentially perform a one-dimensional search for the successor of a and answer range maxima queries at each step. Let T_x denote the compact range tree on the x-coordinates of points. T_x is implemented as in variant (i) of Lemma 1; hence, we can find the interval $[c_v, d_v]$ for any node v in $O(\log^{\varepsilon} n)$ time. We also store a compact structure for range maximum queries $M(v)$ in every node v: given a range $[i, j]$, $M(v)$ returns the index $i \le t \le j$ of the point p with the greatest x-coordinate in $S(v)[i..j]$. We also store a structure for range minimum queries $M'(v)$. $M(v)$ and $M'(v)$ use $O(n)$ bits and answer queries in $O(1)$ time [9]. Hence all $M(u)$ and $M'(u)$ for $u \in T_x$ use $O(n)$ space. Finally, an $O(n)$ space level ancestor structure enables us to find the depth-d ancestor of any node $u \in T_x$ in $O(1)$ time [2].

Let π denote the search path for a in the tree T_x: π connects the root of T_x with the leaf that contains the smallest value $a_x \ge a$. Our procedure looks for the lowest node v_f on π such that $S(v) \cap Q \ne \emptyset$. For simplicity we assume that the length of π is a power of 2. We initialize v_l to the leaf that contains a_x; we initialize v_u to the root node. The node v_f is found by a binary search on π. We say that a node w is the middle node between u and v if w is on the path from u to v and the length of the path from u to w equals to the length of the path from w to v. We set the node v_m to be the middle node between v_u and v_l. Then we find the index t_m of the maximal element in $S(v_m)[c_{v_m}..d_{v_m}]$ and the point $p_m = S(v_m)[t_m]$. If $p_m.x \ge a$, then v_f is either v_m or its descendant; hence, we set $v_u = v_m$. If $p_m.x < a$, then v_f is an ancestor of v_m; hence, we set $v_l = v_m$. The search procedure continues until v_u is the parent of v_m. Finally, we test nodes v_u and v_l and identify v_f (if such v_f exists).

Fact 1. *If the child v' of v_f belongs to π, then v' is the left child of v_f.*

Proof: Suppose that v' is the right child of v_f and let v'' be the sibling of v'. By definition of v_f, $Q \cap S(v') = \emptyset$. Since v' belongs to π and v'' is the left child, $p.x < a$ for all points $p \in S(v'')$. Since $S(v_f) = S(v') \cup S(v'')$, $Q \cap S(v_f) = \emptyset$ and we obtain a contradiction. □

Since $v' \in \pi$ is the left child of v_f, $p.x \geq a$ for all $p \in S(v'')$ for the sibling v'' of v. Moreover, $p.x < a$ for all points $p \in S(v')[c_{v'}, d_{v'}]$ by definition of v_f. Therefore the range successor is the point with minimal x-coordinate in $S(v'')[c_{v''}..d_{v''}]$.

The search procedure visits $O(\log\log n)$ nodes and spends $O(\log^{\varepsilon} n)$ time in each node, thus the total query time is $O(\log^{\varepsilon} n \log\log n)$. By replacing $\varepsilon' < \varepsilon$ in the above construction, we obtain the following result.

Lemma 2. *There exists a data structure that uses $O(n)$ space and answers orthogonal range successor queries in $O(\log^{\varepsilon} n)$ time.*

If we use the compact tree that needs $\Theta(n \log\log n)$ space, then $g(n) = \log\log n$. Using the same structure as in the proof of Lemma 2, we obtain the following.

Lemma 3. *There exists a data structure that uses $O(n \log\log n)$ space and answers orthogonal range successor queries in $O((\log\log n)^2)$ time.*

Sorted Reporting Queries. We can answer sorted reporting queries by answering a sequence of range successor queries. Consider a query $Q = [a, b] \times [c, d]$. Let p_1 be the answer to the range successor query $Q_1 = [a, +\infty] \times [c, d]$. For $i \geq 2$, let p_i be the answer to the query $Q_i = [p_{i-1}.x, +\infty] \times [c, d]$. The sequence of points $p_1, \ldots p_k$ is the sequence of k leftmost points in $[a, b] \times [c, d]$ sorted by their x-coordinates. We observe that our procedure also works in the *online modus* when k is not known in advance. That is, we can output the points of $Q \cap S$ in the left-to-right order until the procedure is stopped by the user or all points in $Q \cap S$ are reported.

Theorem 1. *There exist a data structures that uses $O(n)$ space and answer sorted range reporting queries in $O((k+1)\log^{\varepsilon} n)$ time, and that use $O(n \log\log n)$ space and answer those queries in $O((k+1)(\log\log n)^2)$ time.*

4 Three-Sided Reporting in Optimal Time

In this section we present optimal time data structures for two special cases of sorted two-dimensional queries. In the first part of this section we describe a data structure that answers sorted one-sided queries: for a query c we report all points p, $p.y \leq c$, sorted in increasing order of their x-coordinates. Then we will show how to answer three-sided queries, i.e., to report all points p, $a \leq p.x \leq b$ and $p.y \leq c$, sorted in increasing order by their x-coordinates.

One-Sided Sorted Reporting. We start by describing a data structure that answers queries in $O(\log n + k)$ time; our solution is based on a standard range tree decomposition of the query interval $[1, c]$ into $O(\log n)$ intervals. Then we show how to reduce the query time to $O(k + \log\log n)$. This improvement uses an additional data structure for the case when $k \leq \log n$ points must be reported.

We construct a range tree on the y-coordinates. For every node $v \in T$, the list $L(v)$ contains all points that belong to v sorted by their x-coordinates. Suppose that we want to return k points p with smallest x-coordinates such that $p.y \leq c$.

We can represent the interval $[1, c]$ as a union of $O(\log n)$ node ranges for nodes $v_i \in T$. The search procedure visits each v_i and finds the leftmost point (that is, the first point) in every list $L(v_i)$. Those points are kept in a data structure D. Then we repeat the following step k times: We find the leftmost point p stored in D, output p and remove it from D. If p belongs to a list $L(v_i)$, we find the point p' that follows p in $L(v_i)$ and insert p' into D. As D contains $O(\log n)$ points, we support updates and find the leftmost point in D in $O(1)$ time [10]. Hence, we can initialize D in $O(\log n)$ time and then report k points in $O(k)$ time.

We can reduce the query time to $O(k + \log\log n)$ by constructing additional data structures. If $k \geq \log n$ the data structure described above already answers a query in $O(k + \log n) = O(k)$ time. The case $k \leq \log n$ can be handled as follows. We store for each $p \in S$ a list $V(p)$. Among all points $p' \in S$ such that $p'.y \leq p.y$ the list $V(p)$ contains $\log n$ points with the smallest x-coordinates. Points in $V(p)$ are sorted in increasing order by their x-coordinates. To find k leftmost points in $[1, c]$ for $k < \log n$, we identify the highest point $p_c \in S$ such that $p_c.y \leq c$ and report the first k points in $V(p_c)$. The point p_c can be found in $O(\log\log n)$ time using the van Emde Boas data structure [20]. If p_c is known, then a query can be answered in $O(k)$ time for any value of k.

One last improvement will be important for the data structure of Lemma 5. Let S_m denote the set of $\lceil \log\log n \rceil$ lowest points in S. We store the y-coordinates of $p \in S_m$ in the q-heap F. Using F, we can find the highest $p_m \in S_m$, such that $p_m.y \leq c$, in $O(1)$ time [10]. Let $n_c = |\{\, p \in S \,|\, p.y \leq c \,\}|$. If $n_c \leq \log\log n$, then $p_m = p_c$. As described above, we can answer a query in $O(k)$ time when p_c is known. Hence, a query can be answered in $O(k)$ time if $n_c \leq \log\log n$.

Lemma 4. *There exists an $O(n \log n)$ space data structure that supports one-sided sorted range reporting queries in $O(\log\log n + k)$ time. If the highest point p with $p.y \leq c$ is known, then one-sided sorted queries can be supported in $O(k)$ time. If $|\{\, p \in S \,|\, p.y \leq c \,\}| \leq \log\log n$, a sorted range reporting query $[1, c]$ can be answered in $O(k)$ time.*

Three-Sided Sorted Queries. We construct a range tree on x-coordinates of points. For any node v, the data structure $D(v)$ of Lemma 4 supports one-sided queries on $S(v)$ as described above. For each root-to-leaf path π we store two data structures, $R_1(\pi)$ and $R_2(\pi)$. Let π^+ and π^- be defined as follows. If v belongs to a path π and v is the left child of its parent, then its sibling v' belongs to π^+. If v belongs to π and v is the right child of its parent, then its sibling v' belongs to π^-. The data structure $R_1(\pi)$ contains the lowest point in $S(v')$ for each $v' \in \pi^+$; if $v \in \pi$ is a leaf, $R_1(\pi)$ also contains the point stored in v. The data structure $R_2(\pi)$ contains the lowest point in $S(v')$ for each $v' \in \pi^-$; if $v \in \pi$ is a leaf, $R_2(\pi)$ also contains the point stored in v. Let $lev(v)$ denote the level of a node v (the level of a node v is the length of the path from the root to v). If a point $p \in R_i(\pi)$, $i = 1, 2$, comes from a node v, then $lev(p) = lev(v)$. For a given query (c, l) the data structure $R_1(\pi)$ ($R_2(\pi)$) reports points p such that $p.y \leq c$ and $lev(p) \geq l$ sorted in decreasing (increasing) order by $lev(p)$. Since a data structure $R_i(\pi)$, $i = 1, 2$, contain $O(\log n)$ points, the point with the k-th largest (smallest) value of $lev(p)$ among all p with $p.y \leq c$ can be found in $O(1)$ time. The implementation of structures $R_i(\pi)$ is based on standard bit techniques and will be described in the full version.

Consider a query $Q = [a, b] \times [1, c]$. Let π_a and π_b be the paths from the root to a and b respectively. Suppose that the lowest node $v \in \pi_a \cap \pi_b$ is situated on level $lev(v) = l$. Then all points p such that $p.x \in [a, b]$ belong to some node v such that $v \in \pi_a^+$ and $lev(v) > l$ or $v \in \pi_b^-$ and $lev(v) > l$. We start by finding the leftmost point p in $R_1(\pi_a)$ such that $lev(p) > l$ and $p.y \le c$. Since the x-coordinates of points in $R_1(\pi_a)$ decrease as $lev(p)$ increases, this is equivalent to finding the point $p_1 \in R_1(\pi_a)$ such that $p_1.y \le c$ and $lev(p_1)$ is maximal. If $lev(p_1) > l$, we visit the node $v_1 \in \pi_a^+$ that contains p_1; using $D(v_1)$, we report the k leftmost points $p' \in S(v_1)$ such that $p'.y \le c$. Then, we find the point p_2 with the next largest value of $lev(p)$ among all $p \in R_1(\pi_a)$ such that $p.y \le c$; we visit the node $v_2 \in \pi_a^+$ that contains p_2 and proceed as above. The procedure continues until k points are output or there are no more points $p \in R_1(\pi_a)$, $lev(p) > l$ and $p.y \le c$. If $k' < k$ points were reported, we visit selected nodes $u \in \pi_b^-$ and report remaining $k - k'$ points using a symmetric procedure.

Let k_i denote the number of reported points from the set $S(v_i)$ and let $m_i = Q \cap S(v_i)$. We spend $O(k_i)$ time in a visited node v_i if $k_i \ge \log\log n$ or $m_i < \log\log n$. If $k_j < \log\log n$ and $m_j \ge \log\log n$, then we spend $O(\log\log n + k_j)$ time in the respective node v_j. Thus we spend $O(\log\log n + k_j)$ time in a node v_j only if $m_j > k_j$, i.e., only if not all points from $S(v_j) \cap Q$ are reported. Since at most one such node v_j is visited, the total time needed to answer all one-sided queries is $O(\sum_i k_i + \log\log n) = O(\log\log n + k)$.

Lemma 5. *There exists an $O(n \log^2 n)$ space data structure that answers three-sided sorted reporting queries in $O(\log\log n + k)$ time.*

Online queries. We assumed in Lemmas 4 and 5 that parameter k is fixed and given with the query. Our data structures can also support queries in the online modus using the method originally described in [3]. The main idea is that we find roughly $\Theta(k_i)$ leftmost points from the query range for $k_i = 2^i$ and $i = 1, 2, \ldots$; while k_i points are reported, we simultaneously compute the following $\Theta(k_{i+1})$ points in the background. For a more extensive description, refer to [18, Section 4.1], where the same method for a slightly different problem is described.

5 Two-Dimensional Range Reporting in Optimal Time

We store points in a compact range tree T_y on y-coordinates. We use the variant (iii) of Lemma 1 that uses $O(n \log^\varepsilon n)$ space and retrieves the coordinates of the r-th point from $S(v)$ in $O(1)$ time. Moreover, the sets $S(v)$, $v \in T_y$, are divided into groups $G_i(v)$. Each $G_i(v)$, except of the last one, contains $\lceil \log^3 n \rceil$ points. For $i < j$, each point assigned to $G_i(v)$ has smaller x-coordinate than any point in $G_j(v)$. The set $S'(v)$ contains selected elements from $S(v)$. If v is the right child of its parent, then $S'(v)$ contains $\lceil \log\log n \rceil$ points with smallest y-coordinates from each group $G_i(v)$; structure $D'(v)$ supports three-sided sorted queries of the form $[a, b] \times [0, c]$ on points of $S'(v)$. If v is the left child of its parent, then $S'(v)$

contains $\lceil \log\log n \rceil$ points with largest y-coordinates from each group $G_i(v)$; data structure $D'(v)$ supports three-sided sorted queries of the form $[a,b] \times [c,+\infty]$ on points of $S'(v)$. For each point $p' \in S'(v)$ we store the index i of the group $G_i(v)$ that contains p. We also store the point with the largest x-coordinate from each $G_i(v)$ in a structure $E(v)$ that supports $O(\log\log n)$ time searches [20].

For all points in each group $G_i(v)$ we store an array $A_i(v)$ that contains points sorted by their y-coordinates. Each point is specified by the rank of its x-coordinate in $G_i(v)$; so each entry uses $O(\log\log n)$ bits of space.

To answer a query $Q = [a,b] \times [c,d]$, we find the lowest common ancestor v_c of the leaves that contain c and d. Let v_l and v_r be the left and the right children of v_c. All points in $Q \cap S$ belong to either $([a,b] \times [c,+\infty]) \cap S(v_l)$ or $([a,b] \times [0,d]) \cap S(v_r)$. We generate the sorted list of k leftmost points in $Q \cap S$ by merging the lists of k leftmost points in $([a,b] \times [c,+\infty]) \cap S(v_l)$ and $([a,b] \times [0,d]) \cap S(v_r)$. Thus it suffices to answer sorted three-sided queries $([a,b] \times [c,+\infty])$ and $([a,b] \times [0,d])$ in nodes v_l and v_r respectively.

We consider a query $([a,b] \times [0,d]) \cap S(v_r)$; query $[a,b] \times [c,+\infty]$ is answered symmetrically. Assume $[a,b]$ fits into one group $G_i(v_r)$, i.e., all points p such that $a \le p.x \le b$ belong to one group $G_i(v_r)$. We can find the y-rank d_r of the highest point $p \in G_i(v_r)$, such that $p.y \le d$ in $O(\lg\lg n)$ time by binary search in $A_i(v_r)$. Let a_r and b_r be the ranks of a and b in $G_i(v_r)$. We can find the positions of k leftmost points in $([a_r,b_r] \times [0,d_r]) \cap G_i(v_r)$ using a data structure $H_i(v_r)$. $H_i(v_r)$ contains the y-ranks and x-ranks of points in $G_i(v_r)$ and answers sorted three-sided queries on $G_i(v_r)$. By Lemma 5, $H_i(v_r)$ uses $O(|G_i(v_r)|(\log\log n)^3)$ bits and supports queries in $O(\log\log\log n + k)$ time. Actual coordinates of points can be obtained from their ranks in $G_i(v_r)$ in $O(1)$ time per point: if the x-rank of a point is known, we can compute its position in $S(v_r)$; we obtain x-coordinates of the i-th point in $S(v_r)$ using variant (iii) of Lemma 1.

Now assume $[a,b]$ spans several groups $G_i(v_r), \ldots, G_j(v_r)$ for $i < j$. That is, the x-coordinates of all points in groups $G_{i+1}(v_r), \ldots, G_{j-1}(v_r)$ belong to $[a,b]$; the x-coordinate of at least one point in $G_i(v_r)$ ($G_j(v_r)$) is smaller than a (larger than b) but the x-coordinate of at least one point in $G_i(v_r)$ and $G_j(v_r)$ belongs to $[a,b]$. Indices i and j are found in $O(\log\log n)$ time using $E(v_r)$. We report at most k leftmost points in $([a,b] \times [0,d]) \cap G_i(v_r)$ just as described above.

Let $k_1 = |([a,b] \times [0,d]) \cap G_i(v_r)|$; if $k_1 \ge k$, the query is answered. Otherwise, we report $k' = k - k_1$ leftmost points in $([a,b] \times [0,d]) \cap (G_{i+1}(v_r) \cup \ldots \cup G_{j-1}(v_r))$ using the following method. Let a' and b' be the minimal and the maximal x-coordinates of points in $G_{i+1}(v_r)$ and $G_{j-1}(v_r)$, respectively. The main idea is to answer the query $Q' = ([a',b'] \times [0,d]) \cap S'(v_r)$ in the online modus using the data structure $D'(v_r)$. If some group $G_t(v_r)$, $i < t < j$, contains less than $\lceil \log\log n \rceil$ points p with $p.y \le d$, then all such p belong to $S'(v_r)$ and will be reported by $D'(v_r)$. Suppose that $D'(v_r)$ reported $\log\log n$ points that belong to the same group $G_t(v_r)$. Then we find the rank d_t of d among the y-coordinates of points in $G_t(v_r)$. Using $H_t(v_r)$, we report the positions of all points $p \in G_t(v_r)$, such that the rank of $p.y$ in $G_t(v_r)$ is at most d_t, in the left-to right order; we can also identify the coordinates of every such p in $O(1)$ time per point. The query to $H_t(v_r)$ is terminated when all such points are reported or when the total number of reported points is k.

We need $O(\log\log n + k_t)$ time to answer a query on $H_t(v_r)$, where k_t denotes the number of reported points from $G_t(v_r)$. Let $m_t = |Q' \cap G_t(v_r)|$ If G_t is the last examined group, then $k_t \leq m_t$; otherwise $k_t = m_t$. We send a query to $G_t(v_r)$ only if $G_t(v_r)$ contains at least $\log\log n$ points from Q'. Hence, a query to $G_t(v_r)$ takes $O(\log\log n + k_t) = O(k_t)$ time, unless $G_t(v_r)$ is the last examined group. Thus all necessary queries to $G_t(v_r)$ for $i<t<j$ take $O(\log\log n + k)$ time.

Finally, if the total number of points in $([a,b] \times [0,d]) \cap (G_i(v_r) \cup \ldots \cup G_{j-1}(v_r))$ is smaller than k, we also report the remaining points from $([a,b] \times [0,d]) \cap G_j(v_r)$.

The compact tree T_y uses $O(n \log^{\varepsilon} n)$ words of space. A data structure $D'(v)$ uses $O(|S'(v)| \log^2 n \log\log n) = O(|S(v)| \log\log n / \log n)$ words of space. Since all sets $S(v)$, $v \in T_y$, contain $O(n \log n)$ points, all $D'(v)$ use $O(n \log\log n)$ words of space. A data structure for a group $G_i(v)$ uses $O(|G_i(v)|(\log\log n)^3)$ bits. Since all $G_i(v)$ for all $v \in T_y$ contain $O(n \log n)$ elements, data structures for all groups $G_i(v)$ use $O(n(\log\log n)^3)$ words of $\log n$ bits.

Theorem 2. *There exists a $O(n \log^{\varepsilon} n)$ space data structure that answers two-dimensional sorted reporting queries in $O(\log\log n + k)$ time.*

6 Applications

In this section we will describe data structures for several indexing and computational geometry problems. A text (string) T of length n is pre-processed and stored in a data structure so that certain queries concerning some substrings of T can be answered efficiently.

Preliminaries. In a suffix tree $\mathcal{T}$ for a text T, every leaf of $\mathcal{T}$ is associated with a suffix of T. If the leaves of $\mathcal{T}$ are listed from left to right, then the corresponding suffixes of T are lexicographically sorted. For any pattern P, we can find in $O(|P|)$ time the special node $v \in \mathcal{T}$, called the *locus* of P. The starting position of every suffix in the subtree of $v = \mathsf{locus}(P)$ is the location of an occurrence of P. We define the rank of a suffix Suf as the number of T's suffixes that are lexicographically smaller than or equal to Suf. The ranks of all suffixes in $v = \mathsf{locus}(P)$ belong to an interval $[\mathit{left}(P), \mathit{right}(P)]$, where $\mathit{left}(P)$ and $\mathit{right}(P)$ denote the ranks of the leftmost and the rightmost suffixes in the subtree of v. Thus for any pattern P there is a unique range $[\mathit{left}(P), \mathit{right}(P)]$; pattern P occurs at position i in T if and only if the rank of suffix $T[i..n]$ belongs to $[\mathit{left}(P), \mathit{right}(P)]$. Refer to [13] for a more extensive description of suffix trees and related concepts.

We will frequently use a special set of points, further called *the position set for T*. Every point p in the position set corresponds to a unique suffix Suf of a string T; the y-coordinate of p equals to the rank of Suf and the x-coordinate of p equals to the starting position of Suf in T.

Successive List Indexing. In this problem a query consists of a pattern P and an index j, $1 \leq j \leq n$. We want to find the first (leftmost) occurrence of P at position

$i \geq j$. A successive list indexing query (P, j) is equivalent to finding the point p from the position set such that p belongs to the range $[j, n] \times [left(P), right(P)]$ and the x-coordinate of p is minimal. Thus a list indexing query is equivalent to a range successor query on the position set. Using Theorems 1 and 2 to answer range successor queries, we obtain the following result.

Corollary 1. *We can store a string T in an $O(nf(n))$ space data structure, so that for any pattern P and any index j, $1 \leq j \leq n$, the leftmost occurrence of P at position $i \geq j$ can be found in $O(g(n))$ time for (i) $f(n) = O(1)$ and $g(n) = O(\log^{\varepsilon} n)$; (ii) $f(n) = O(\log\log n)$ and $g(n) = O((\log\log n)^2)$; (iii) $f(n) = O(\log^{\varepsilon} n)$ and $g(n) = O(\log\log n)$.*

Range Non-Overlapping Indexing. In the string statistics problem we want to find the maximum number of non-overlapping occurrences of a pattern P. In [14] the *range non-overlapping indexing problem* was introduced: instead of just computing the maximum number of occurrences we want to find the longest sequence of non-overlapping occurrences of P. It was shown [14] that the range non-overlapping indexing problem can be solved via k successive list indexing queries; here k denotes the maximal number of non-overlapping occurrences.

Corollary 2. *The range non-overlapping indexing problem can be solved in $O(|P| + kg(n))$ time with an $O(nf(n))$ space data structure, where $g(n)$ and $f(n)$ are defined as in Corollary 1.*

Other, more far-fetched applications, are described next.

6.1 Pattern Matching with Variable-Length Don't Cares

We must determine whether a query pattern $P = P_1 * P_2 * P_3 \ldots * P_m$ occurs in T. The special symbol $*$ is the Kleene star symbol; it corresponds to an arbitrary sequence of (zero or more) characters from the original alphabet of T. The parameter m can be specified at query time. In [22] the authors showed how to answer such queries in $O(\sum_{i=1}^{m} |P_i|)$ and $O(n)$ space in the case when the alphabet size is $\log^{O(1)} n$. In this paper we describe a data structure for an arbitrarily large alphabet. Using the approach of [22], we can reduce such a query for P to answering m successive list indexing queries. First, we identify the leftmost occurrence of P_1 in T by answering the successive list indexing query $(P_1, 1)$. Let j_1 denote the leftmost position of P_1. $P_1 * P_2 * P_3 \ldots * P_m$ occurs in T if and only if $P_2 * P_3 \ldots * P_m$ occurs at position $i \geq j_1 + |P_1|$. We find the leftmost occurrence $j_2 \geq j_1 + |P_1|$ of P_2 by answering the query $(P_2, j_1 + |P_1|)$. $P_2 * P_3 \ldots * P_m$ occurs in T at position $i_2 \geq j_1 + |P_1|$ if and only if $P_3 * P_m$ occurs at position $i_3 \geq j_2 + |P_2|$. Proceeding in the same way we find the leftmost possible positions for $P_4 * \ldots * P_m$. Thus we answer m successive list indexing queries (P_t, i_t), $t = 1, \ldots, m$; here $i_1 = 1$, $i_t = j_{t-1} + |P_{i-1}|$ for $t \geq 2$, and j_{t-1} denotes the answer to the $(t-1)$-th query.

Corollary 3. *We can determine whether a text T contains a substring $P = P_1 * \ldots P_{m-1} * P_m$ in $O(\sum_{i=1}^{m} |P_i| + mg(n))$ time using an $O(nf(n))$ space data structure, where $g(n)$ and $f(n)$ are defined as in Corollaries 1 and 2.*

6.2 Ordered Substring Searching

Suppose that a data structure contains a text T and we want to report occurrences of a query pattern P in the left-to-right order, i.e., in the same order as they appear in T; in some case we may want to find only the k leftmost occurrences. In this section we describe two solutions for this problem. Then we show how sorted range reporting can be used to solve the position-restricted variant of this problem. We denote by occ the number of P's occurrences in T that are reported when a query is answered.

Data Structure with Optimal Query Time. Such queries can be answered in $O(|P| + \mathrm{occ})$ time and $O(n)$ space using the suffix tree and the data structure of Brodal *et al.* [3]. Positions of suffixes are stored in lexicographic order in the suffix array A; the k-th entry $A[k]$ contains the starting position of the k-th suffix in the lexicographic order. In [3] the authors described an $O(n)$ space data structure that answers online sorted range reporting queries: for any $i \geq j$, we can report in $O(j - i + 1)$ time all entries $A[t]$, $i \leq t \leq j$, sorted in increasing order by their values. Occurrences of a pattern P can be reported in the left-to-right order as follows. Using a suffix tree, we find $\mathit{left}(P)$ and $\mathit{right}(P)$ in $O(|P|)$ time. Then we report all suffixes in the interval $[\mathit{left}(P), \mathit{right}(P)]$ sorted by their starting positions using the data structure of [3] on A.

Corollary 4. *We can answer a sorted substring matching query in $O(|P|+\mathrm{occ})$ time using a $O(n)$ space data structure.*

Succinct Data Structure. The space usage of a data structure for sorted pattern matching can be further reduced. We store a compressed suffix array for T and a succinct data structure for range minimum queries. We use the implementation of the compressed suffix array described in [12] that needs $(1 + 1/\varepsilon)nH_k + o(n)$ bits for $\sigma = \log^{O(1)} n$, where σ denotes the alphabet size and H_k is the k-th order entropy. Using the results of [12], we can find the position of the i-th lexicographically smallest suffix in $O(\log^{\varepsilon} n)$ time. We can also find $\mathit{left}(P)$ and $\mathit{right}(P)$ for any P in $O(|P|)$ time. We also store the range minimum data structure [9] for the array A defined above. For any $i \leq j$, we can find such $k = \mathsf{rmq}(i, j)$ that $A[k] \leq A[t]$ for any $i \leq t \leq j$. Observe that A itself is not stored; we only store the structure from [9] that uses $O(n)$ bits of space. Now occurrences of P are reported as follows. An initially empty queue Q contains suffix positions; with every suffix position p we also store an interval $[l_p, r_p]$ and the rank i_p of the suffix that starts at position p. Let $l = \mathit{left}(P)$ and $r = \mathit{right}(P)$. We find $i_f = \mathsf{rmq}(l, r)$ and the position p_f of the suffix with rank i_f. The position p_f with its rank i_f and the associated interval $[l, r]$ is inserted into Q. We repeat the following steps until Q is empty. The item with the minimal value of p_t is extracted from Q. Let i_t and $[l_t, r_t]$ denote the rank and interval stored with p_t. We answer queries $i' = \mathsf{rmq}(l_t, i_t - 1)$ and $i'' = \mathsf{rmq}(i_t + 1, r_t)$ and identify the positions p', p'' of suffixes with ranks i', i''. Finally, we insert items $(p', i', [l_t, i_t - 1])$ and $(p'', i'', [i_t + 1, r_t])$ into Q. Using the van Emde Boas data structure, we can implement each operation on Q in $O(\log \log n)$ time. We can find the position of a suffix with rank i in $O(\log^{\varepsilon} n)$ time. Thus the total time that we need to answer a query is $O(|P| + \text{occ} \log^{\varepsilon} n)$. Our data structure uses $(1 + 1/\varepsilon)nH_k + O(n)$ bits. We observe however that we need $O(\text{occ} \log n)$ additional bits at the time when a query is answered.

Corollary 5. *If the alphabet size $\sigma = \log^{O(1)} n$, then we can answer an ordered substring searching query in $O(|P|+\text{occ} \log^{\varepsilon} n)$ time using a $(1+1/\varepsilon)nH_k+O(n)$-bit data structure.*

Position-Restricted Ordered Substring Searching. The position restricted substring searching problem was introduced by Mäkinen and Navarro in [16]. Given a range $[i, j]$ we want to report all occurrences of P that start at position t, $i \le t \le j$. If we want to report occurrences of P at positions from $[i, j]$ in the sorted order, then this is equivalent to answering a sorted range reporting query $[i, j] \times [\mathit{left}(P), \mathit{right}(P)]$. Hence, we can obtain the same time-space trade-offs as in Theorems 1 and 2.

6.3 Maximal Points in a 2D Range and Rectangular Visibility

A point p *dominates* another point q if $p.x \ge q.x$ and $p.y \ge q.y$. A point $p \in S$ is *a maximal point* if p is not dominated by any other point $q \in S$. In a two-dimensional maximal points range query, we must find all maximal points in $Q \cap S$ for a query rectangle Q. We refer to [4] and references therein for description of previous results.

We can answer such queries using orthogonal range successor queries. For simplicity, we assume that all points have different x- and y-coordinates. Suppose that maximal points in the range $Q = [a, b] \times [c, d]$ must be listed. For $i \ge 1$, we report a point p_i such that $p_i.x \ge p.x$ for any $p \in Q_{i-1} \cap S$, where $Q_0 = Q$ and $Q_j = [a, p_i.x] \times [p_i.y, d]$ for $j \ge 1$. Our reporting procedure is completed when $Q_i \cap S = \emptyset$. Clearly, finding a point p_i or determining that no such p_i exists is equivalent to answering a range successor query for Q_{i-1}. Thus we can find k maximal points in $O(kg(n))$ time using an $O(nf(n))$ space data structure, where $g(n)$ and $f(n)$ are again defined as in Corollary 1.

A point $p \in S$ is rectangularly visible from a point q if $Q_{pq} \cap S = \emptyset$, where Q_{pq} is the rectangle with points p and q at its opposite corners. In the rectangle visibility problem, we must determine all points $p \in S$ that are visible from a query point q. Rectangular visibility problem is equivalent to finding maximal points in $Q \cap S$ for $Q = [0, q.x] \times [0, q.y]$. Hence, we can find points rectangularly visible from q in $O(kg(n))$ time using an $O(nf(n))$ space data structure.

References

1. Agarwal, P., Erickson, J.: Geometric range searching and its relatives. Contemporary Mathematics 223, 1–56 (1999)
2. Bender, M.A., Farach-Colton, M.: The level ancestor problem simplified. Theor. Comput. Sci. 321(1), 5–12 (2004)

3. Brodal, G.S., Fagerberg, R., Greve, M., López-Ortiz, A.: Online Sorted Range Reporting. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 173–182. Springer, Heidelberg (2009)
4. Brodal, G.S., Tsakalidis, K.: Dynamic Planar Range Maxima Queries. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 256–267. Springer, Heidelberg (2011)
5. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the RAM, revisited. In: Proc. 27th SoCG, pp. 1–10 (2011)
6. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput. 17(3), 427–462 (1988)
7. Crochemore, M., Iliopoulos, C.S., Kubica, M., Rahman, M.S., Walen, T.: Improved algorithms for the range next value problem and applications. In: Proc. 25th STACS, pp. 205–216 (2008)
8. Crochemore, M., Iliopoulos, C.S., Rahman, M.S.: Finding Patterns in Given Intervals. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 645–656. Springer, Heidelberg (2007)
9. Fischer, J.: Optimal Succinctness for Range Minimum Queries. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 158–169. Springer, Heidelberg (2010)
10. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. 48(3), 533–551 (1994)
11. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proc. 16th STOC, pp. 135–143 (1984)
12. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proc. 14th SODA, pp. 841–850 (2003)
13. Gusfield, D.: Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press (1997)
14. Keller, O., Kopelowitz, T., Lewenstein, M.: Range Non-overlapping Indexing and Successive List Indexing. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 625–636. Springer, Heidelberg (2007)
15. Lenhof, H.-P., Smid, M.H.M.: Using persistent data structures for adding range restrictions to searching problems. RAIRO Theor. Inf. and Appl. 28(1), 25–49 (1994)
16. Mäkinen, V., Navarro, G.: Rank and select revisited and extended. Theor. Comput. Sci. 387(3), 332–347 (2007)
17. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. 57(1), 37–49 (1998)
18. Navarro, G., Nekrich, Y.: Top-k document retrieval in optimal time and linear space. In: Proc. 22nd SODA, pp. 1066–1077 (2012)
19. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proc. 38th STOC, pp. 232–240 (2006)
20. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Math. Sys. Theory 10, 99–127 (1977)
21. Yu, C.-C., Hon, W.-K., Wang, B.-F.: Improved data structures for the orthogonal range successor problem. Comput. Geom. Theory Appl. 44, 148–159 (2011)
22. Yu, C.-C., Wang, B.-F., Kuo, C.-C.: Efficient Indexes for the Positional Pattern Matching Problem and Two Related Problems over Small Alphabets. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 13–24. Springer, Heidelberg (2010)

String Indexing for Patterns with Wildcards

Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind

Technical University of Denmark, DTU Informatics
{phbi,ilg,hwvi}@imm.dtu.dk

Abstract. We consider the problem of indexing a string t of length n to report the occurrences of a query pattern p containing m characters and j wildcards. Let *occ* be the number of occurrences of p in t, and σ the size of the alphabet. We obtain the following results.

- A linear space index with query time $O(m + \sigma^j \log\log n + occ)$. This significantly improves the previously best known linear space index by Lam et al. [ISAAC 2007], which requires query time $\Theta(jn)$ in the worst case.
- An index with query time $O(m + j + occ)$ using space $O(\sigma^{k^2} n \log^k \log n)$, where k is the maximum number of wildcards allowed in the pattern. This is the first non-trivial bound with this query time.
- A time-space trade-off, generalizing the index by Cole et al. [STOC 2004].

Our results are obtained using a novel combination of well-known and new techniques, which could be of independent interest.

1 Introduction

The *string indexing problem* is to build an index for a string t such that the occurrences of a query pattern p can be reported. The classic suffix tree data structure [31] combined with perfect hashing [15] gives a linear space solution for string indexing with optimal query time, i.e., an $O(n)$ space data structure that supports queries in $O(m + occ)$ time, where *occ* is the number of occurrences of p in t.

Recently, various extensions of the classic string indexing problem that allow errors or wildcards (also known as gaps or don't cares) have been studied [11,21,29,28,6,25,26]. In this paper, we focus on one of the most basic of these extensions, namely, *string indexing for patterns with wildcards*. In this problem, only the pattern contains wildcards, and the goal is to report all occurrences of p in t, where a wildcard is allowed to match any character in t.

String indexing for patterns with wildcards finds several natural applications in large-scale data processing areas such as information retrieval, bioinformatics, data mining, and internet traffic analysis. For instance in bioinformatics, the PROSITE data base [18,5] supports searching for protein patterns containing wildcards.

Despite significant interest in the problem and its many variations, most of the basic questions remain unsolved. We introduce three new indexes and obtain several new bounds for string indexing with wildcards in the pattern. If the index can handle patterns containing an unbounded number of wildcards, we call it

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 283–294, 2012.

an *unbounded wildcard index*, otherwise we refer to the index as a *k-bounded wildcard index*, where k is the maximum number of wildcards allowed in p. Let n be the length of the indexed string t, and σ be the size of the alphabet. We define m and j to be the number of characters and wildcards in p, respectively. Consequently, the length of p is $m + j$.

Previous Work. Exact string matching has been generalized with error bounds in many different ways. In particular, allowing matches within a bounded hamming or edit distance, known as approximate string matching, has been subject to much research [22,23,27,12,10,29,6,25,11,26,16,2]. Another generalization was suggested by Fischer and Paterson [14], allowing wildcards in the text or pattern.

Work on the wildcard problem has mostly focused on the non-indexing variant, where the string t is not preprocessed in advance [14,13,9,20,8,4]. Some solutions to the indexing problem consider the case where wildcards appear only in the indexed string [28] or in both the string and the pattern [11,21].

In the following, we summarize the known indexes that support wildcards in the pattern only. We focus on the case where $k > 1$, since for $k = 0$ the problem is classic string indexing. For $k = 1$, Cole et al. [11] describe a selection of specialized solutions. However, these solutions do not generalize to larger k.

Several simple solutions to the problem exist for $k > 1$. Using a suffix tree T for t [31], we can find all occurrences of p in a top-down traversal starting from the root. When we reach a wildcard character in p in location $\ell \in T$, the search branches out, consuming the first character on all outgoing edges from ℓ. This gives an unbounded wildcard index using $O(n)$ space with query time $O(\sigma^j m + occ)$, where occ is the total number of occurrences of p in t. Alternatively, we can build a compressed trie storing all possible modifications of all suffixes of t containing at most k wildcards. This gives a k-bounded wildcard index using $O(n^{k+1})$ space with query time $O(m + j + occ)$.

In 2004, Cole et al. [11] gave an elegant k-bounded wildcard index using $O(n \log^k n)$ space and with $O(m + 2^j \log\log n + occ)$ query time. For sufficiently small values of j this significantly improves the previous bounds. The key components in this solution are a new data structure for *longest common prefix (LCP) queries* and a *heavy path decomposition* [17] of the suffix tree for the text t. Given a pattern p, the LCP data structure supports efficiently inserting all suffixes of p into the suffix tree for t, such that subsequent longest common prefix queries between any pair of suffixes from t and p can be answered in $O(\log\log n)$ time. This is the $\log\log n$ term in the query time. The heavy path decomposition partitions the suffix tree into disjoint *heavy paths* such that any root-to-leaf path contains at most a logarithmic number of heavy paths. Cole et al. [11] show how to reduce the size of the index by only creating additional wildcard tries for the off-path subtries. This leads to the $O(n \log^k n)$ space bound. Secondly, using the new tries, the top-down search branches at most twice for each wildcard, leading to the 2^j term in the query time. Though Cole et al. [11] did not consider unbounded wildcard indexes, the technique can be extended to this case by using only the LCP data structure. This leads to an unbounded wildcard index with query time $O(m + \sigma^j \log\log n + occ)$ using space $O(n \log n)$.

The solutions described by Cole et al. [11] all have bounds, which are exponential in the number of wildcards in the pattern. Very recently, Lewenstein [24] used similar techniques to improve the bounds to be exponential in the number of *gaps* in the pattern (a gap is a maximal substring of consecutive wildcards). Assuming that the pattern contains at most g gaps each of size at most G, Lewenstein obtains a bounded index with query time $O(m + 2^{\gamma} \log\log n + occ)$ using space $O(n(G^2 \log n)^g)$, where $\gamma \leq g$ is the number of gaps in the pattern.

A different approach was taken by Iliopoulos and Rahman [19], who describe an unbounded wildcard index using linear space. For a pattern p consisting of strings $p_0, p_1, \ldots, p_j$ (subpatterns) interleaved by j wildcards, the query time of the index is $O(m + \sum_{i=0}^{j} occ(p_i, t))$, where $occ(p_i, t)$ denotes the number of matches of p_i in t. This was later improved by Lam et al. [21] with an index that determines complete matches by first identifying potential matches of the subpatterns in t and subsequently verifying each possible match for validity using interval stabbing on the subpatterns. Their solution is an unbounded wildcard index with query time $O\left(m + j \min_{0 \leq i \leq j} occ(p_i, t)\right)$ using linear space. However, both of these solutions have a worst case query time of $\Theta(jn)$, since there may be $\Theta(n)$ matches for a subpattern, but no matches of p.

The unbounded wildcard index by Iliopoulos and Rahman [19] was the first index to achieve query time linear in m while using $O(n)$ space. Recently, Chan et al. [6] considered the related problem of obtaining a k-mismatch index supporting queries in time linear in m and using $O(n)$ space. They describe an index with a query time of $O(m + (\log n)^{k(k+1)} \log\log n + occ)$. However, this bound assumes a constant-size alphabet and a constant number of errors. In this paper we make no assumptions on the size of these parameters.

Our Results. Our main contributions are three new wildcard indexes.

Theorem 1. *Let t be a string of length n from an alphabet of size σ. There is an unbounded wildcard index for t using $O(n)$ space. The index can report the occurrences of a pattern with m characters and j wildcards in time $O(m + \sigma^j \log\log n + occ)$.*

Compared to the solution by Cole et al. [11], we obtain the same query time while reducing the space usage by a factor $\log n$. We also significantly improve upon the previously best known linear space index by Lam et al. [21], as we match the linear space usage while improving the worst-case query time from $\Theta(jn)$ to $O(m + \sigma^j \log\log n + occ)$ provided $j \leq \log_\sigma n$. Our solution is faster than the simple suffix tree index for $m = \Omega(\log\log n)$. Thus, for sufficiently small j we improve upon the previously known unbounded wildcard indexes.

The main idea of the solution is to combine an ART decomposition [1] of the suffix tree for t with the LCP data structure. The suffix tree is decomposed into a number of logarithmic-sized bottom trees and a single top tree. We introduce a new variant of the LCP data structure for use on the bottom trees, which supports queries in logarithmic time and linear space. The logarithmic size of the bottom trees leads to LCP queries in time $O(\log\log n)$. On the top tree we use the LCP data structure by Cole et al. [11] to answer queries in time $O(\log\log n)$. The number of LCP queries performed during a search for p is $O(\sigma^j)$, yielding the $\sigma^j \log\log n$ term in the query time. The reduced size of the top tree causes the index to be linear in size.

Theorem 2. *Let t be a string of length n from an alphabet of size σ. For $2 \leq \beta < \sigma$, there is a k-bounded wildcard index using $O(n \log(n) \log_\beta^{k-1} n)$ space. The index can report the occurrences in t of a pattern with m characters and $j \leq k$ wildcards in time $O(m + \beta^j \log\log n + occ)$.*

The theorem provides a time-space trade-off for k-bounded wildcard indexes. Compared to the index by Cole et al. [11], we reduce the space usage by a factor $\log^{k-1} \beta$ by increasing the branching factor from 2 to β. For $\beta = 2$ the index is identical to the index by Cole et al. The result is obtained by generalizing the wildcard index described by Cole et al. We use a *heavy α-tree decomposition*, which is a new technique generalizing the classic heavy path decomposition by Harel and Tarjan [17]. This decomposition could be of independent interest. We also show that for $\beta = 1$ the same technique yields an index with query time $O(m + j + occ)$ using space $O(nh^k)$, where h is the height of the suffix tree for t.

Theorem 3. *Let t be a string of length n from an alphabet of size σ. There is a k-bounded wildcard index for t using $O(\sigma^{k^2} n \log^k \log n)$ space. The index can report the occurrences of a pattern with m characters and $j \leq k$ wildcards in time $O(m + j + occ)$.*

To our knowledge this is the first linear time index with a non-trivial space bound. The result improves upon the space usage of the simple linear time index when $\sigma^k < n/\log\log n$. To achieve this result, we use the $O(nh^k)$ space index to obtain a black-box reduction that can produce a linear time index from an existing index. The idea is to build the $O(nh^k)$ space index with support for short patterns, and query another index if the pattern is long. This technique is closely related to the concept of *filtering search* introduced by Chazelle [7] and has previously been applied for indexing problems [3,6]. The theorem follows from applying the black-box reduction to the index of Theorem 1.

Our three indexes also support searching for query patterns with *variable length gaps*, i.e., patterns of the form $p = p_0 *\{a_1, b_1\}\, p_1 *\{a_2, b_2\} \ldots *\{a_j, b_j\}\, p_j$, where $*\{a_i, b_i\}$ denotes a variable length gap that matches an arbitrary substring of length between a_i and b_i, both inclusive. We will not consider variable length gaps in this paper.

We have left out some proofs due to lack of space, and we refer the reader to [30], which also treats query patterns with variable length gaps.

2 Preliminaries

We introduce the following notation. Let $p = p_0 * p_1 * \ldots * p_j$ be a pattern consisting of $j + 1$ strings $p_0, p_1, \ldots, p_j \in \Sigma^*$ (subpatterns) interleaved by $j \leq k$

wildcards. The substring starting at position $l \in \{1,\ldots,n\}$ in t is an occurrence of p if and only if each subpattern p_i matches the corresponding substring in t. We define $t[i,j] = \varepsilon$ for $i > j$, $t[i,j] = t[1,j]$ for $i < 1$ and $t[i,j] = t[i,|t|]$ for $j > |t|$. Furthermore $m = \sum_{r=0}^{j} |p_r|$ is the number of characters in p, and we assume without loss of generality that $m > 0$ and $k > 0$.

Let $\mathrm{pref}_i(t) = t[1,i]$ and $\mathrm{suff}_i(t) = t[i,n]$ denote the prefix and suffix of t of length i and $n-i+1$, respectively. Omitting the subscripts, we let $\mathrm{pref}(t)$ and $\mathrm{suff}(t)$ denote the set of all non-empty prefixes and suffixes of t, respectively. We extend the definitions of prefix and suffix to sets of strings $S \subseteq \Sigma^*$ as follows.

$$\mathrm{pref}_i(S) = \{\mathrm{pref}_i(x) \mid x \in S\} \qquad \mathrm{suff}_i(S) = \{\mathrm{suff}_i(x) \mid x \in S\}$$

$$\mathrm{pref}(S) = \bigcup_{x \in S} \mathrm{pref}(x) \qquad \mathrm{suff}(S) = \bigcup_{x \in S} \mathrm{suff}(x)$$

A set of strings S is *prefix-free* if no string in S is a prefix of another string in S. Any string set S can be made prefix-free by appending the same unique character $\$ \notin \Sigma$ to each string in S.

Trees and Tries. For a tree T, the root is denoted $\mathrm{root}(T)$, while $\mathrm{height}(T)$ is the number of edges on a longest path from $\mathrm{root}(T)$ to a leaf of T. A compressed trie $T(S)$ is a tree storing a prefix-free set of strings $S \subset \Sigma^*$. The edges are labeled with substrings of the strings in S, such that a path from the root to a leaf corresponds to a unique string of S. All internal vertices (except the root) have at least two children, and all labels on the outgoing edges of a vertex have different initial characters.

A *location* $\ell \in T(S)$ may refer to either a vertex or a position on an edge in $T(S)$. Formally, $\ell = (v,s)$ where v is a vertex in $T(S)$ and $s \in \Sigma^*$ is a prefix of the label on an outgoing edge of v. If $s = \varepsilon$, we also refer to ℓ as an *explicit vertex*, otherwise ℓ is called an *implicit vertex*. There is a one-to-one mapping between locations in $T(S)$ and unique prefixes in $\mathrm{pref}(S)$. The prefix $x \in \mathrm{pref}(S)$ corresponding to a location $\ell \in T(S)$ is obtained by concatenating the edge labels on the path from $\mathrm{root}(T(S))$ to ℓ. Consequently, we use x and ℓ interchangeably, and we let $|\ell| = |x|$ denote the length of x. Since S is assumed prefix-free, each leaf of $T(S)$ is a string in S, and conversely. The *suffix tree* for t denotes the compressed trie over all suffixes of t, i.e., $T(\mathrm{suff}(t))$. We define $T_\ell(S)$ as the subtrie of $T(S)$ rooted at ℓ. That is, $T_\ell(S)$ contains the suffixes of strings in $T(S)$ starting from ℓ. Formally, $T_\ell(S) = T(S_\ell)$, where

$$S_\ell = \left\{\mathrm{suff}_{|\ell|}(x) \mid x \in S \wedge \mathrm{pref}_{|\ell|}(x) = \ell\right\} .$$

Heavy Path Decomposition. For a vertex v in a rooted tree T, we define $\mathrm{weight}(v)$ to be the number of leaves in T_v, where T_v denotes the subtree rooted at v. We define $\mathrm{weight}(T) = \mathrm{weight}(\mathrm{root}(T))$. The *heavy path decomposition* of T, introduced by Harel and Tarjan [17], classifies each edge as either *light* or *heavy*. For each vertex $v \in T$, we classify the edge going from v to its child of maximum weight (breaking ties arbitrarily) as heavy. The remaining edges are light.

This construction has the property that on a path from the root to any vertex, $O(\log(\text{weight}(T)))$ heavy paths are traversed. For a heavy path decomposition of a compressed trie $T(S)$, we assume that the heavy paths are extended such that the label on each light edge contains exactly one character.

3 The LCP Data Structure

Cole et al. [11] introduced the the *Longest Common Prefix (LCP) data structure*, which provides a way to traverse a compressed trie without tracing the query string one character at a time. In this section we give a brief, self-contained description of the data structure and show a new property that is essential for obtaining Theorem 1.

The LCP data structure stores a collection of compressed tries $T(C_1), T(C_2), \dots, T(C_q)$ over the string sets $C_1, C_2, \dots, C_q \subset \Sigma^*$. Each C_i is a set of substrings of the indexed string t. The purpose of the LCP data structure is to support LCP queries

LCP(x, i, ℓ): Returns the location in $T(C_i)$ where the search for the string $x \in \Sigma^*$ stops when starting in location $\ell \in T(C_i)$.

If ℓ is the root of $T(C_i)$, we refer to the above LCP query as a *rooted LCP query*. Otherwise the query is called an *unrooted LCP query*. In addition to the compressed tries $T(C_1), \dots, T(C_q)$, the LCP data structure also stores the suffix tree for t, denoted $T(C)$ where $C = \text{suff}(t)$. The following lemma is implicit in the paper by Cole et al. [11].

Lemma 1 (Cole et al.). *Provided x has been preprocessed in time $O(|x|)$, the LCP data structure can answer rooted LCP queries on $T(C_i)$ for any suffix of x in time $O(\log\log|C|)$ using space $O(|C| + \sum_{i=1}^{q} |C_i|)$. Unrooted LCP queries on $T(C_i)$ can be performed in time $O(\log\log|C|)$ using $O(|C_i|\log|C_i|)$ additional space.*

We extend the LCP data structure by showing that support for slower unrooted LCP queries on a compressed trie $T(C_i)$ can be added using linear additional space.

Lemma 2. *Unrooted LCP queries on $T(C_i)$ can be performed in time $O(\log|C_i| + \log\log|C|)$ using $O(|C_i|)$ additional space.*

Proof. We initially create a heavy path decomposition for all compressed tries $T(C_1), \dots, T(C_q)$. The search path for x starting in ℓ traverses a number of heavy paths in $T(C_i)$. Intuitively, an unrooted LCP query can be answered by following the $O(\log|C_i|)$ heavy paths that the search path passes through. For each heavy path, the next heavy path can be identified in constant time. On the final heavy path, a predecessor query is needed to determine the exact location where the search path stops.

For a heavy path H, we let h denote the distance which the search path for x follows H. Cole et al. [11] showed that h can be determined in constant time by performing nearest common ancestor queries on $T(C)$. To answer $\textsc{lcp}(x, i, \ell)$ we identify the heavy path H of $T(C_i)$ that ℓ is part of and compute the distance h as described by Cole et al. If x leaves H on a light edge, indexing distance h into H from ℓ yields an explicit vertex v. At v, a constant time lookup for $x[h + 1]$ determines the light edge on which x leaves H. Since the light edge has a label of length one, the next location ℓ' on that edge is the root of the next heavy path. We continue the search for the remaining suffix of x from ℓ' recursively by a new unrooted LCP query $\textsc{lcp}(\mathrm{suff}_{h+2}(x), i, \ell')$. If H is the heavy path on which the search for x stops, the location at distance h (i.e., the answer to the original LCP query) is not necessarily an explicit vertex, and may not be found by indexing into H. In that case a predecessor query for h is performed on H to determine the preceding explicit vertex and thereby the location $\textsc{lcp}(x, i, \ell)$. Answering an unrooted LCP query entails at most $\log |C_i|$ recursive steps, each taking constant time. The final recursive step may require a predecessor query taking time $O(\log \log |C|)$. Consequently, an unrooted LCP query can be answered in time $O(\log |C_i| + \log \log |C|)$ using $O(|C_i|)$ additional space to store the predecessor data structures for each heavy path. □

4 An Unbounded Wildcard Index Using Linear Space

In this section we show how to obtain Theorem 1 by applying an ART decomposition on the suffix tree for t and storing the top and bottom trees in the LCP data structure.

ART Decomposition. The ART decomposition introduced by Alstrup et al. [1] decomposes a tree into a single *top tree* and a number of *bottom trees*. The construction is defined by two rules:

1. A bottom tree is a subtree rooted in a vertex of minimal depth such that the subtree contains no more than χ leaves.
2. Vertices that are not in any bottom tree make up the top tree.

The decomposition has the following key property.

Lemma 3 (Alstrup et al.). *The ART decomposition with parameter χ for a rooted tree T with n leaves produces a top tree with at most $\frac{n}{\chi+1}$ leaves.*

Obtaining the Index. Applying an ART decomposition on $T(\mathrm{suff}(t))$ with $\chi = \log n$, we obtain a top tree T' and a number of bottom trees $B_1, B_2, \ldots, B_q$ each of size at most $\log n$. From Lemma 3, T' has at most $\frac{n}{\log n}$ leaves and hence $O(\frac{n}{\log n})$ vertices since T' is a compressed trie.

To facilitate the search, the top and bottom trees are stored in an LCP data structure, noting that these compressed tries only contain substrings of t. Using Lemma 2, we add support for unrooted $O(\log \chi + \log \log n) = O(\log \log n)$ time LCP queries on the bottom trees using $O(n)$ additional space in total. For the top tree we apply Lemma 1 to add support for unrooted LCP queries in time

$O(\log\log n)$ using $O(\frac{n}{\log n}\log\frac{n}{\log n}) = O(n)$ additional space. Since the branching factor is not reduced, $O(\sigma^i)$ LCP queries, each taking time $O(\log\log n)$, are performed for the subpattern p_i. This concludes the proof of Theorem 1.

5 A Time-Space Trade-Off for k-Bounded Wildcard Indexes

In this section we will show Theorem 2. We first introduce the necessary constructions.

Heavy α-Tree Decomposition. The *heavy α-tree decomposition* is a generalization of the well-known heavy path decomposition introduced by Harel and Tarjan [17]. The purpose is to decompose a rooted tree T into a number of *heavy trees* joined by light edges, such that a path to the root of T traverses at most a logarithmic number of heavy trees. For use in the construction, we define a proper weight function on the vertices of T, to be a function satisfying $\text{weight}(v) \geq \sum_{w \text{ child of } v} \text{weight}(w)$. Observe that using the number of vertices or the number of leaves in the subtree rooted at v as the weight of v satisfies this property. The decomposition is then constructed by classifying edges in T as being heavy or light according to the following rule. For every vertex $v \in T$, the edges to the α heaviest children of v (breaking ties arbitrarily) are heavy, and the remaining edges are light. Observe that for $\alpha = 1$, this results in a heavy path decomposition. Given a heavy α-tree decomposition of T, we define $\text{lightdepth}_\alpha(v)$ to be the number of light edges on a path from the vertex $v \in T$ to the root of T. The key property of this construction is captured by the following lemma.

Lemma 4. *For any vertex v in a rooted tree T and $\alpha > 0$*

$$\text{lightdepth}_\alpha(v) \leq \log_{\alpha+1} \text{weight}(\text{root}(T))$$

Lemma 4 holds for any heavy α-tree decomposition obtained using a proper weight function on T. In the remaining part of the paper we will assume that the weight of a vertex is the number of leaves in the subtree rooted at v.

We define $\text{lightheight}_\alpha(T)$ to be the maximum light depth of a vertex in T, and remark that $\text{lightheight}_0(T) = \text{height}(T)$. For a vertex v in a compressed trie $T(S)$, we let $\text{lightstrings}(v)$ denote the set of strings starting in one of the light edges leaving v. That is, $\text{lightstrings}(v)$ is the union of the set of strings in the subtries $T_\ell(S)$ where ℓ is the first location on a light outgoing edge of v, i.e., $|\ell| = |v| + 1$.

Wildcard Trees. We introduce the (β, k)*-wildcard tree*, denoted $T^k_\beta(C')$, where $1 \leq \beta < \sigma$ is a chosen parameter. This data structure stores a collection of strings $C' \subset \Sigma^+$ in a compressed trie such that the search for a pattern p with at most k wildcards branches to at most β locations in $T^k_\beta(C')$ when consuming a single wildcard of p. In particular for $\beta = 1$, the search for p never branches and the search time becomes linear in the length of p. For a vertex v, we define the wildcard height of v to be the number of wildcards on the path from v to the root. Intuitively, given a wildcard tree that supports i wildcards, support for an extra wildcard is added by joining a new tree to each vertex v with wildcard height i by an edge labeled $*$. This tree is searched if a wildcard is consumed in v. Formally, $T^k_\beta(C')$ is built recursively as follows.

> **Construction of** $T^i_\beta(S)$: Produce a heavy $(\beta - 1)$-tree decomposition of $T(S)$, then for each internal vertex $v \in T(S)$ join v to the root of $T^{i-1}_\beta(\mathrm{suff}_2(\mathrm{lightstrings}(v))$ by an edge labeled $*$. Let $T^0_\beta(S) = T(S)$.

Wildcard Tree Index. Given a collection C' of strings and a pattern p, we can identify the strings of C' having a prefix matching p by constructing $T^k_\beta(C')$. Searching $T^k_\beta(C')$ is similar to the suffix tree search, except when consuming a wildcard character of p in an explicit vertex $v \in T^k_\beta(C')$ with more than β children. In that case the search branches to the root of the wildcard tree joined to v and to the first location on the $\beta - 1$ heavy edges of v, effectively letting the wildcard match the first character on all edges from v. Consequently, the search for p branches to a total of at most $\sum_{i=0}^{j} \beta^i = O(\beta^j)$ locations, each of which requires $O(m)$ time, resulting in a query time $O(\beta^j m + occ)$. For $\beta = 1$ the query time is $O(m + j + occ)$.

Lemma 5. *For any integer* $1 \le \beta < \sigma$, *the wildcard tree* $T^k_\beta(C')$ *has query time* $O(\beta^j m + j + occ)$. *The wildcard tree stores* $O(|C'|H^k)$ *strings, where* H *is an upper bound on the light height of all compressed tries* $T(S)$ *satisfying* $S \subseteq \mathrm{suff}_d(C')$ *for some integer* d.

Proof. We prove that the total number of strings (leaves) in $T^i_\beta(S)$, denoted $|T^i_\beta(S)|$, is at most $|S| \sum_{j=0}^{i} H^j = O(|S|H^i)$. The proof is by induction on i. The base case $i = 0$ holds, since $T^0_\beta(S) = T(S)$ contains $|S| = |S| \sum_{j=0}^{0} H^j$ strings. For the induction step, assume that $|T^i_\beta(S)| \le |S| \sum_{j=0}^{i} H^j$. Let $S_v = \mathrm{suff}_2(\mathrm{lightstrings}(v))$ for a vertex $v \in T(S)$. From the construction we have that the number of strings in $T^{i+1}_\beta(S)$ is the number of strings in $T(S)$ plus the number of strings in the wildcard trees joined to the vertices of $T(S)$. That is,

$$|T^{i+1}_\beta(S)| \;=\; |S| + \sum_{v \in T(S)} |T^i_\beta(S_v)| \;\overset{IH}{\le}\; |S| + \sum_{v \in T(S)} |S_v| \sum_{j=0}^{i} H^j \;.$$

The string sets S_v consist of suffixes of strings in S. Consider a string $x \in S$, i.e., a leaf in $T(S)$. The number of times a suffix of x appears in a set S_v is equal to the light depth of x in $T(S)$. S is also a set of suffixes of C', and hence H is an upper bound on the maximum light depth of $T(S)$. This establishes that $\sum_{v \in T(S)} |S_v| \;\le\; |S|H$, thus showing that $|T^{i+1}_\beta(S)| \le |S| + |S|H \sum_{j=0}^{i} H^j = |S| \sum_{j=0}^{i+1} H^j$. □

Constructing the wildcard tree $T_\beta^k(C)$, where $C = \mathrm{suff}(t)$, we obtain a wildcard index with the following properties.

Lemma 6. *Let t be a string of length n from an alphabet of size σ. For $2 \le \beta < \sigma$ there is a k-bounded wildcard index for t using $O\big(n \log_\beta^k n\big)$ space. The index can report the occurrences of a pattern with m characters and $j \le k$ wildcards in time $O\left(\beta^j m + occ\right)$.*

Wildcard Tree Index Using the LCP Data Structure. The wildcard index of Lemma 6 reduces the branching factor of the suffix tree search from σ to β, but still has the drawback that the search for a subpattern p_i from a location $\ell \in T_\beta^k(C)$ takes $O(|p_i|)$ time. This can be addressed by combining the index with the LCP data structure as in Cole et al. [11]. In that way, the search for a subpattern can be done in time $O(\log\log n)$. The index is obtained by modifying the construction of $T_\beta^i(S)$ such that each $T(S)$ is added to the LCP data structure prior to joining the $(\beta, i-1)$-wildcard trees to the vertices of $T(S)$. For all $T(S)$ except the final $T(S) = T_\beta^0(S)$, support for unrooted LCP queries in time $O(\log\log n)$ is added using additional $O(|S|\log|S|)$ space. For the final $T(S)$ we only need support for rooted queries. Upon receiving the query pattern $p = p_1 * p_2 * \ldots * p_k$, each p_i is preprocessed in time $O(|p_i|)$ to support LCP queries for any suffix of p_i. The search for p proceeds as described for the normal wildcard tree, except now rooted and unrooted LCP queries are used to search for suffixes of $p_0, p_1, \ldots, p_k$.

In the search for p, a total of at most $\sum_{i=0}^{j} \beta^i = O(\beta^j)$ LCP queries, each taking time $O(\log\log n)$, are performed. Preprocessing $p_0, p_1, \ldots, p_j$ takes $\sum_{i=0}^{j} |p_i| = m$ time, so the query time is $O(m + \beta^j \log\log n + occ)$. The space needed to store the index is $O(n \log_\beta^k n)$ for $T_\beta^k(C)$ plus the space needed to store the LCP data structure.

Adding support for rooted LCP queries requires linear space in the total size of the compressed tries, i.e., $O(n \log_\beta^k n)$. Let $T(S_0), T(S_1), \ldots, T(S_q)$ denote the compressed tries with support for unrooted LCP queries. Since each S_i contains at most n strings and $\sum_{i=0}^{q} |S_i| = |T_\beta^{k-1}(C)|$, by Lemma 1, the additional space required to support unrooted LCP queries is

$$O\big(\sum_{i=0}^{q} |S_i| \log|S_i|\big) = O\big(\log n \sum_{i=0}^{q} |S_i|\big) = O\left(\log n |T_\beta^{k-1}(C')|\right) = O\big(n \log(n) \log_\beta^{k-1} n\big) ,$$

which is an upper bound on the total space required to store the wildcard index. This concludes the proof of Theorem 2. The k-bounded wildcard index described by Cole et al. [11] is obtained as a special case of Theorem 2.

Corollary 1 (Cole et al.). *Let t be a string of length n from an alphabet of size σ. There is a k-bounded wildcard index for t using $O(n \log^k n)$ space. The index can report the occurrences of a pattern with m characters and $j \le k$ wildcards in time $O(m + 2^j \log\log n + occ)$.*

6 A k-Bounded Wildcard Index with Linear Query Time

Consider the k-bounded wildcard index obtained by creating the wildcard tree $T_1^k(\text{suff}(t))$ for t. This index has linear query time, and we can show that the space usage depends of the height of the suffix tree.

Lemma 7. *Let t be a string of length n from an alphabet of size σ. There is a k-bounded wildcard index for t using $O(nh^k)$ space, where h is the height of the suffix tree for t. The index can report the occurrences of a pattern with m characters and j wildcards in time $O(m + j + occ)$.*

In the worst case the height of the suffix tree is close to n, but combining the index with another wildcard index yields a useful black box reduction. The idea is to query the first index if the pattern is short, and the second index if the pattern is long.

Lemma 8. *Let $F \geq m$ and let G be independent of m and j. Given a wildcard index $\mathcal{A}$ with query time $O(F + G + occ)$ and space usage S, there is a k-bounded wildcard index $\mathcal{B}$ with query time $O(F + j + occ)$ and taking space $O(n \min(G, h)^k + S)$, where h is the height of the suffix tree for t.*

Proof. The wildcard index $\mathcal{B}$ consists of $\mathcal{A}$ as well as a special wildcard index $T_1^k(\text{pref}_G(\text{suff}(t)))$ $\mathcal{C}$, which is a wildcard tree with $\beta = 1$ over the set of all substrings of t of length G. G can be used as an upper bound for the light height in Lemma 5, so the space required to store $\mathcal{C}$ is $O(n \min(G, h)^k)$ by using Lemma 7 if $G > h$. A query on $\mathcal{B}$ results in a query on either $\mathcal{A}$ or $\mathcal{C}$. In case $G < F + j$, we query $\mathcal{A}$ and the query time will be $O(F+G+occ) = O(F+j+occ)$. In case $G \geq F + j$, we query $\mathcal{C}$ with query time $O(m+j+occ) = O(F+j+occ)$. In any case the query time of $\mathcal{B}$ is $O(F + j + occ)$. □

Applying Lemma 8 with $F = m$ and $G = \sigma^k \log\log n$ on the unbounded wildcard index from Theorem 1 yields a new k-bounded wildcard index with linear query time using space $O(\sigma^{k^2} n \log^k \log n)$. This concludes the proof of Theorem 3.

References

1. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: Proc. 39th FOCS, pp. 534–543 (1998)
2. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with k mismatches. In: Proc. 11th SODA, pp. 794–803 (2000)
3. Bille, P., Gørtz, I.L.: Substring Range Reporting. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 299–308. Springer, Heidelberg (2011)
4. Bille, P., Li Gørtz, I., Vildhøj, H.W., Wind, D.K.: String Matching with Variable Length Gaps. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 385–394. Springer, Heidelberg (2010)
5. Bucher, P., Bairoch, A.: A generalized profile syntax for biomolecular sequence motifs and its function in automatic sequence interpretation. In: Proc. 2nd ISMB, pp. 53–61 (1994)
6. Chan, H.L., Lam, T.W., Sung, W.K., Tam, S.L., Wong, S.S.: A linear size index for approximate pattern matching. J. Disc. Algorithms 9(4), 358–364 (2011)
7. Chazelle, B.: Filtering search: A new approach to query-answering. SIAM J. Comput. 15(3), 703–724 (1986)

8. Chen, G., Wu, X., Zhu, X., Arslan, A., He, Y.: Efficient string matching with wildcards and length constraints. Knowl. Inf. Sys. 10(4), 399–419 (2006)
9. Clifford, P., Clifford, R.: Simple deterministic wildcard matching. Inf. Process. Lett. 101(2), 53–54 (2007)
10. Coelho, L.P., Oliveira, A.L.: Dotted Suffix Trees A Structure for Approximate Text Indexing. In: Crestani, F., Ferragina, P., Sanderson, M. (eds.) SPIRE 2006. LNCS, vol. 4209, pp. 329–336. Springer, Heidelberg (2006)
11. Cole, R., Gottlieb, L., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proc. 36th STOC, pp. 91–100 (2004)
12. Cole, R., Hariharan, R.: Approximate string matching: A simpler faster algorithm. In: Proc. 9th SODA, pp. 463–472 (1998)
13. Cole, R., Hariharan, R.: Verifying candidate matches in sparse and wildcard matching. In: Proc. 34rd STOC, pp. 592–601 (2002)
14. Fischer, M.J., Paterson, M.S.: String-Matching and Other Products. In: Complexity of Computation, SIAM-AMS Proceedings, pp. 113–125 (1974)
15. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a Sparse Table with O(1) Worst Case Access Time. J. ACM 31, 538–544 (1984)
16. Galil, Z., Giancarlo, R.: Improved string matching with k mismatches. ACM SIGACT News 17(4), 52–54 (1986)
17. Harel, D., Tarjan, R.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)
18. Hofmann, K., Bucher, P., Falquet, L., Bairoch, A.: The PROSITE database, its status in 1999. Nucleic Acids Res. 27(1), 215–219 (1999)
19. Iliopoulos, C.S., Rahman, M.S.: Pattern matching algorithms with don't cares. In: Proc. 33rd SOFSEM, pp. 116–126 (2007)
20. Kalai, A.: Efficient pattern-matching with don't cares. In: Proc. 13th SODA, pp. 655–656 (2002)
21. Lam, T.-W., Sung, W.-K., Tam, S.-L., Yiu, S.-M.: Space Efficient Indexes for String Matching with Don't Cares. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 846–857. Springer, Heidelberg (2007)
22. Landau, G., Vishkin, U.: Efficient string matching with k mismatches. Theoret. Comput. Sci. 43, 239–249 (1986)
23. Landau, G., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms 10(2), 157–169 (1989)
24. Lewenstein, M.: Indexing with Gaps. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 135–143. Springer, Heidelberg (2011)
25. Maas, M., Nowak, J.: Text indexing with errors. J. Disc. Algorithms 5(4), 662–681 (2007)
26. Navarro, G., Baeza-Yates, R., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. IEEE Data Eng. Bull. 24(4), 19–27 (2001)
27. Sahinalp, S., Vishkin, U.: Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: Proc. 37th FOCS, pp. 320–328 (1996)
28. Tam, A., Wu, E., Lam, T.-W., Yiu, S.-M.: Succinct Text Indexing with Wildcards. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 39–50. Springer, Heidelberg (2009)
29. Tsur, D.: Fast index for approximate string matching. J. Disc. Algorithms 8(4), 339–345 (2010)
30. Vildhøj, H.W., Vind, S.: String Indexing for Patterns with Wildcards. Master's thesis, Technical University of Denmark (2011), http://www.imm.dtu.dk/~hwvi/
31. Weiner, P.: Linear pattern matching algorithms. In: Proc. 14th SWAT, pp. 1–11 (1973)

Linear-Space Data Structures for Range Minority Query in Arrays*

Timothy M. Chan[1], Stephane Durocher[2],
Matthew Skala[2], and Bryan T. Wilkinson[1]

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
{tmchan,b3wilkin}@uwaterloo.ca

[2] Department of Computer Science, University of Manitoba, Winnipeg, Canada
{durocher,skala}@cs.umanitoba.ca

Abstract. We consider range queries in arrays that search for low-frequency elements: least frequent elements and α-minorities. An α-minority of a query range has multiplicity no greater than an α fraction of the elements in the range. Our data structure for the least frequent element range query problem requires $O(n)$ space, $O(n^{3/2})$ preprocessing time, and $O(\sqrt{n})$ query time. A reduction from boolean matrix multiplication to this problem shows the hardness of simultaneous improvements in both preprocessing time and query time. Our data structure for the α-minority range query problem requires $O(n)$ space, supports queries in $O(1/\alpha)$ time, and allows α to be specified at query time.

1 Introduction

The *frequency* of an element x in a multiset stored as an array $A[0 : n-1]$, denoted $\text{freq}_A(x)$, is the number of occurrences (i.e., the multiplicity) of x in A. Given $\alpha \in [0,1]$, an element x is an *α-minority* in A if $1 \leq \text{freq}_A(x) \leq \alpha n$, whereas x is an *α-majority* if $\text{freq}_A(x) > \alpha n$.

We examine two problems which involve preprocessing a given array A to construct a data structure that can efficiently find low-frequency elements in query ranges. A least frequent element range query specifies a pair of indices (i,j) and returns a least frequent element that occurs in $A[i : j]$. An α-minority range query specifies some $\alpha \in [0,1]$ and a pair of indices (i,j), and returns an element whose frequency in $A[i : j]$ is at least 1 and at most $\alpha|j-i+1|$. If no such element exists, the query must not return any element. Whenever we discuss a data structure with a parameter β instead of α, β is fixed before preprocessing. We do so to differentiate from the more challenging case in which different parameter values can be specified at query time.

Several recent results examine the minimum, selection (including median), mode (i.e., the most frequent element), β-majority, and α-majority range query problems on arrays (e.g., [1–3, 5, 7–14, 16–18]). Most relevant to our low-frequency query problems are results for their high-frequency analogues: an

* Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 295–306, 2012.

$O(n)$-space data structure that supports range mode queries in $O(\sqrt{n/\log n})$ time [5], an $O(n \log(1/\beta + 1))$-space data structure that supports β-majority range queries in $O(1/\beta)$ time [9], and a $O(n \log n)$-space data structure that supports α-majority range queries in $O(1/\alpha)$ time [11]. Related generalizations include examinations of the the β-majority range query problem in the dynamic setting [10] and the α-majority range query problem in two dimensions [11]. Greve et al. [13] give a lower bound of $\Omega(\log n / \log(s \cdot w/n))$ on the range mode query time for any data structure that uses s memory cells of w bits in the cell probe model; they show the same bound applies to the problem of determining whether any element in a given query range has frequency exactly k, for any k given at query time. Consequently, no $O(n)$-space data structure can support constant-time (independent of α) α-minority queries.

Our low-frequency query problems have significant differences when compared to their high-frequency analogues. For example, for any (i, j), the frequencies of respective modes of $A[i : j]$ and $A[i : j+1]$ differ by either zero or one. The frequency of the mode of a set increases monotonically with the addition of new elements into the set. Conversely, the frequencies of respective least frequent elements of $A[i : j]$ and $A[i : j+1]$ can differ by any value in $\{i-j, \ldots, 0, 1\}$. Similarly, if x is a mode of $A[i : k]$ and $A[k+1 : j]$, then x is a mode of $A[i : j]$, whereas the analogous property does not hold for least frequent elements.

In Section 2 we consider the least frequent element range query problem. We describe an $O(n)$-space data structure that identifies a least frequent element in a query range in $O(\sqrt{n})$ time. This data structure is a variant of a previous data structure of Chan et al. [5] for the range mode problem (which in turn was an improvement of a previous data structure of Krizanc et al. [16]). In addition, using an argument similar to that of Chan et al. [5], we present a reduction from boolean matrix multiplication to the least frequent element range query problem, showing the hardness of simultaneously improving our preprocessing and query time bounds.

Section 3 contains the main result of this paper: an $O(n)$-space data structure that supports α-minority range queries in $O(1/\alpha)$ time. Our technique is quite different from the previous techniques of Durocher et al. [9] for β-majority range queries and Gagie et al. [11] for α-majority range queries, which have worse space bounds ($O(n \log(1/\beta + 1)$ and $O(n \log n)$, respectively).

In Section 4 we apply a variation of our technique to give an $O(n \log n)$-space data structure that supports α-majority range queries in $O(1/\alpha)$ time. These space and time bounds match those achieved by a recent α-majority data structure of Gagie et al. [11].

Both our data structures in Sections 3 and 4 make interesting use of existing tools from computational geometry. Notably, we apply Chazelle's *hive graphs* [6], which were designed for a seemingly unrelated two-dimensional searching problem: preprocess a set of horizontal line segments so that we can report segments intersecting a given vertical line segment or ray.

2 Finding a Least Frequent Element

2.1 $O(\sqrt{n})$-Time Data Structure

In this section we present an $O(n)$-space data structure that identifies a least frequent element in a query range in $O(\sqrt{n})$ time and requires $O(n^{3/2})$ preprocessing time. Specifically, we will prove the following theorem that implies the above result when $s = \sqrt{n}$:

Theorem 1. *Given an array $A[0 : n-1]$ and any fixed $s \in [1, n]$, there exists an $O(n + s^2)$-space data structure that supports least frequent range query on A in $O(n/s)$ time and requires $O(ns)$ preprocessing time.*

Preprocessing. Given an arbitrary input array $A[0 : n-1]$, we begin by building an array $B[0 : n-1]$ such that $B[x]$ is the rank of $A[x]$ amongst the distinct elements of A. We find the ranks of all the elements by sorting A. Thus, all elements in B are in the range $\{0, \ldots, \Delta - 1\}$, where Δ denotes the number of distinct elements in A. Furthermore, $B[x]$ is a least frequent element in $B[i : j]$ if and only if $A[x]$ is a least frequent element in $A[i : j]$, for any i, j, and x. Following Krizanc et al. [16] and Chan et al. [5], for each $x \in \{0, \ldots, \Delta - 1\}$, we define an array Q_x such that $Q_x[k]$ stores the index of the kth instance of x in B. Since each element in B is represented exactly once in $Q_0, \ldots, Q_{\Delta-1}$, the total space required by $Q_0, \ldots, Q_{\Delta-1}$ is $\Theta(n)$. We also define a rank array $B'[0 : n-1]$ such that for all b, $B'[b]$ denotes the rank (i.e., the index) of b in $Q_{B[b]}$. Therefore, $Q_{B[b]}[B'[b]] = b$. Using these arrays, Chan et al. observe the following lemma (which follows by comparing $Q_{B[i]}[B'[i] + q - 1]$ with j):

Lemma 1 (Chan et al. [5, Lemma 3]). *Given an array $B[0 : n-1]$, there exists an $O(n)$-space data structure that determines in $O(1)$ time for any $0 \leq i \leq j \leq n-1$ and any q whether $B[i : j]$ contains at least q instances of element $B[i]$.*

We also require the following lemma:

Lemma 2. *Given an array $B[0 : n-1]$, there exists an $O(n)$-space data structure that computes in $O(j - i + 1)$ time for any $0 \leq i \leq j \leq n-1$ the frequencies of all elements in $B[i : j]$. In particular, a least frequent element in $B[i : j]$ and its frequency can be computed in $O(j - i + 1)$ time.*

Proof. No actual preprocessing is necessary other than initializing an array $C[0 : \Delta - 1]$ to zero. The query algorithm is similar to counting sort: compute a frequency table for $B[i : j]$ stored in C (i.e., for every x, $C[x]$ corresponds to the frequency of x in $B[i : j]$), then find a minimum element in C. The time required to find the minimum is bounded by $O(j-i+1)$ by comparing all frequencies $C[x]$, where x corresponds to an element in $B[i : j]$ (these are exactly the elements of C that have non-zero values). This procedure is repeated after identifying the minimum to reset C to zero. □

We divide the array B into s blocks of size $t = \lceil n/s \rceil$. A query range $B[i:j]$ spans between 0 and s complete blocks. Let the *span* of $B[i:j]$ be the sequence of complete blocks contained within $B[i:j]$. Let the *prefix* and *suffix* of $B[i:j]$ be the elements of $B[i:j]$ that respectively precede and succeed the span of $B[i:j]$. We precompute the following data for each possible span S:

i. an element of minimum frequency and its frequency in S, among all elements in S, and
ii. an element of minimum frequency and its frequency in S, among all elements (if any) that appear in S but not in the blocks immediately adjacent to the left and right of S.

Since $s(s+1)/2$ spans are possible, these data can be stored in a table D of size $\Theta(s^2)$. We construct this table in $O(ns)$ time by repeatedly passing through the entire array, starting at each of the s block boundaries. We will use the following lemma:

Lemma 3. *There exists a data structure maintaining an initially empty multiset S of elements from $\{0, \ldots, \Delta - 1\}$. It requires $O(\Delta)$ space and preprocessing time and supports the following operations:*

- *Insert(S, e): Inserts element e into multiset S in $O(1)$ time.*
- *LeastFrequentElement(S, k): Returns the k least frequent elements in S, along with their frequencies, in $O(k)$ time.*

Proof. We construct a doubly-linked list L, where each node contains a frequency f and a doubly-linked sublist of all distinct elements with frequency f. The nodes of L are sorted in the ascending order of frequency. Nodes for the sublists are taken from an array $N[0 : \Delta - 1]$ of nodes for each distinct element. Each of these sublist nodes contains a pointer to its containing sublist. It can be verified that an insertion of an element e causes only local changes around $N[e]$ that run in $O(1)$ time. To find the k least frequent elements, we simply iterate through L and its sublists until we have reported k elements or there are no more elements to report. □

During each pass we incrementally build a multiset using the data structure of Lemma 3. At every block boundary (i.e., every t elements) we obtain the least frequent element of the multiset in $O(1)$ time. We must also find the least frequent element excluding the elements contained in two blocks. This set of excluded elements has size $O(t)$ and so the element for which we are searching must appear amongst the $O(t)$ least frequent elements of the multiset, which we can find in $O(t)$ time. The total cost of a single pass is thus $O(n + st) = O(n)$ time. Therefore, the s passes altogether require $O(ns)$ time.

Query Algorithm. Consider arbitrary indices $0 \leq i \leq j \leq n - 1$ and the corresponding query range $R = B[i : j]$. If the prefix and suffix are empty, then the query can be answered in $O(1)$ time by referring to table D. By Lemma 2, if $j - i + 1 < 2t$, then the range query can be answered in $O(t) = O(n/s)$ time.

Now consider the case $j - i + 1 \geq 2t$. In this case, the span, denoted S, must be non-empty. We denote the prefix by P_1 and the suffix by P_2. Let P_1' and P_2' denote the respective blocks that contain P_1 and P_2. We now treat R, S, P_1, P_2, P_1', P_2' as multisets. Let P denote the union of P_1 and P_2. Similarly, let P' denote the union of P_1' and P_2'. We partition the elements of R into four groups (see Figure 1) and find an element of minimum frequency among those in each group:

Case 1. elements of R that are in P but not S,
Case 2. elements of R that are in S and P,
Case 3. elements of R that are in S and P', but not P, and
Case 4. elements of R that are in S but not P'.

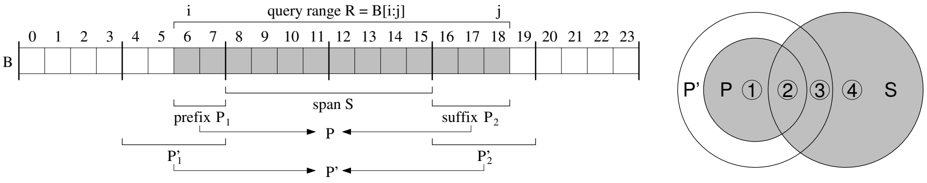


Fig. 1. Every element in the query range R (shaded) is in P or S, partitioned into sets 1–4

We first show how to determine which elements of P' fall into Cases 1, 2, and 3. It suffices to determine for each element of P' whether or not the element appears in P and whether or not the element appears in S. We determine which elements appear in P by simply iterating through P. To determine which elements appear in S, we first find the closest occurrence of each element to S in a scan through P'. Assume that we have one such closest element $B[x]$ at index x. Assume without loss of generality that it appears in P_1'. The next occurrence of element $B[x]$ is at index $x' = Q_{B[x]}[B'[x] + 1]$, which we compute in $O(1)$ time. Thus, S contains an occurrence of element $B[x]$ if and only if x' lies inside S.

The least frequent element in R is given by the least frequent of those found in each of the four cases defined above:

Case 1. By Lemma 2, we compute the frequencies of all elements in P_1 in $O(t)$ time, omitting the final step of resetting the frequency table to zero. We then repeat for P_2 so that the frequency table contains aggregate data for all of P. Consider all elements that occur in P but not in S. For each such element e, $\text{freq}_R(e) = \text{freq}_P(e)$. So, the least frequent of these elements in R is the element with minimum non-zero entry in the frequency table.

Case 2. Let f denote the precomputed minimum frequency of any element in S, which is stored in table D. The minimum frequency in R of any element present in both S and P is at least f and at most $f + 2t$. For each element e that occurs in both S and P_1, we find the leftmost occurrence of e within P_1 in a scan through P_1. We repeat in a symmetric fashion in P_2. Then, by Lemma 1, we can check in $O(1)$ time whether an element e in both S and P has frequency in R less than some threshold. We begin with a threshold of $f + 2t + 1$. If an element e has frequency less than the threshold, we find its actual frequency by iterating through Q_e (forward or backwards depending on whether we are considering an element in P_1 or P_2) until reaching an index within R. This frequency becomes our new threshold. We repeat with all other elements that occur in both S and P. The last element to change the threshold is the least frequent of these elements. Since the threshold can decrease to at most f, the total time spent finding exact frequencies is $O(t)$.

Case 3. Consider all elements that occur in both S and P' but not in P. As in Case 2, their frequencies in R are bounded between f and $f + 2t$. We can thus apply the same technique as in Case 2. However, for each element, instead of finding the leftmost occurrence in P_1 or the rightmost occurrence in P_2 from which to base the queries of Lemma 1, we find the rightmost occurrence in P'_1 or the leftmost occurrence in P'_2.

Case 4. Consider all elements that occur in S but not in P'. For each such element e, $\text{freq}_R(e) = \text{freq}_S(e)$. The least frequent of these elements has been precomputed and can be found in table D in $O(1)$ time.

Analysis. In addition to the arrays A, B, and B' (each $O(n)$ space), the data structure stores the tables $Q_0, \ldots, Q_{\Delta-1}$ ($O(n)$ total space), the tables D ($O(s^2)$ space), and a frequency table ($O(\Delta) \subseteq O(n)$ space). Populating, scanning, and resetting the frequency table during a query requires $O(t) = O(n/s)$ time. The query algorithm involves a constant number of scans of the blocks P'_1 and P'_2. Each element is processed in $O(1)$ amortized time, resulting in $O(t)$ total time. Thus, the data structure has space $O(n + s^2)$ and supports queries in $O(t) = O(n/s)$ time in the worst case. This completes the proof of Theorem 1.

2.2 Reduction from Boolean Matrix Multiplication

We follow the technique of Chan et al. [5] to multiply two $n \times n$ boolean matrices L and R via least frequent element range queries. In particular, we build an array A of size $n' \in O(n^2)$, and after preprocessing the array in $P(n')$ time we perform n^2 least frequent element queries, each in $Q(n')$ time, to calculate $M = LR$. The result is Theorem 2.

Theorem 2. *Given a data structure for least frequent element query in an array of n elements with $P(n)$ preprocessing time and $Q(n)$ query time, there exists an algorithm for boolean matrix multiplication of two $n \times n$ matrices that runs in $O(P(n^2) + n^2 Q(n^2))$ time.*

Thus, a data structure for least frequent element with $P(n) \in o(n^{3/2-\epsilon})$ and with $Q(n) \in o(n^{1/2-\epsilon})$ would yield an algorithm for boolean matrix multiplication that runs in $o(n^{3-2\epsilon})$ time, via purely combinatorial means.

The technique of Chan et al. [5] first reduces boolean matrix multiplication to set disjointness queries between sets encoding the rows of L and the columns of

R. Let $U = \{1, \ldots, n\}$ be our ground set. We are left with the following problem: given sets $L_1, \ldots, L_n \subseteq U$ and $R_1, \ldots, R_n \subseteq U$, determine whether $L_i \cap R_j = \emptyset$ for all $i, j \in \{1, \ldots, n\}$.

Our construction of A involves creating $2n + 1$ blocks of n elements: a block for each set L_i, followed by a block containing each element of U, followed by a block for each set R_j. The block for set L_i contains all elements of L_i followed by all elements of $U \setminus L_i$. The block for set R_j contains all elements of $U \setminus R_j$ followed by all elements of R_j.

We determine whether or not L_i and R_j are disjoint via a single least frequent element query from the leftmost element of $U \setminus L_i$ to the rightmost element of $U \setminus R_j$. This query range contains $i + j - 1 > 0$ complete blocks, each containing some permutation of U. If L_i and R_j are disjoint, then every element of U occurs either in $U \setminus L_i$ or $U \setminus R_j$. Thus, in this case, the least frequent element has frequency greater than $i + j - 1$. If L_i and R_j are not disjoint then some element occurs in neither $U \setminus L_i$ nor $U \setminus R_j$, and thus has the lowest possible frequency of $i + j - 1$. Thus, $L_i \cap R_j = \emptyset$ if and only if the frequency of the least frequent element in the range is exactly $i + j - 1$.

In total we must preprocess A, which has size $O(n^2)$ and perform n^2 least frequent element queries in this array, resulting in an algorithm that requires $O(P(n^2) + n^2 Q(n^2))$ time. This completes the proof of Theorem 2.

3 Range Minority

In this section we describe an $O(n)$-space data structure that identifies an α-minority element, if any exists, in a query range in $O(1/\alpha)$ time. We first reduce this α-minority range query problem to the problem of identifying the leftmost occurrences of the k leftmost distinct elements on or to the right of a given query index. We call the latter problem *distinct element searching* and we require that k can be specified at query time.

Lemma 4. *Given a data structure D for distinct element searching that requires $S_D(n)$ space and $Q_D(n, k)$ query time to report k elements, there exists a data structure for the α-minority range query problem that requires $O(S_D(n) + n)$ space and $O(Q_D(n, 1/\alpha) + 1/\alpha)$ query time.*

Proof. As described in Section 2.1, suppose we store in an array B', for each index i of A, a count of the number of times $A[i]$ occurs previously in A, and for each distinct element $x \in \{0, \ldots, \Delta - 1\}$, a sorted array Q_x of all the indices where x occurs in A. These arrays require $O(n)$ space. By Lemma 1, we can check in $O(1)$ time whether there are at least q instances of $A[i]$ in the range $A[i : j]$ for any $q \geq 0$ and $j \geq i$.

Observe that any element in a range is either an α-majority or an α-minority for the range and fewer than $1/\alpha$ distinct elements can be α-majorities. Thus, if we can find $1/\alpha$ distinct elements in a range, then at least one of them must be an α-minority.

Given a query range $A[i:j]$, we use data structure D to find the leftmost occurrences of the $1/\alpha$ leftmost distinct elements on or to the right of index i in $Q(n, 1/\alpha)$ time. Some of these leftmost occurrences may lie to the right of index j; we can ignore these elements as no occurrence of these elements lies in $A[i:j]$. There are $O(1/\alpha)$ remaining leftmost occurrences of leftmost distinct elements. Consider such an occurrence at index ℓ. Since this is the first occurrence of $A[\ell]$ on or after index i, the frequency of $A[\ell]$ in $A[\ell:j]$ is equal to the frequency of $A[\ell]$ in $A[i:j]$. We can then check whether or not $A[\ell]$ is an α-minority in $A[i:j]$ in $O(1)$ time by setting $q = \alpha(j-i+1)+1$ in Lemma 1. Repeating for all leftmost occurrences requires $O(1/\alpha)$ time.

If we find an α-minority we are done. If we do not find an α-minority, then there must not have been $1/\alpha$ distinct elements to check. In that case, we checked all distinct elements in $A[i:j]$ so there cannot be an α-minority. □

We can now focus on distinct element searching. If all queries use a common fixed k (as is the case if our goal is to solve just the range β-minority problem), there is a simple data structure that requires $O(n)$ space and $O(k)$ query time: for each i that is a multiple of k, store the k leftmost distinct elements to the right of index i; then for an arbitrary index i, we can answer a query by examining the k elements stored at $j' = \lceil i/k \rceil k$ in addition to the $O(k)$ elements in $A[i:j']$. However, it is not obvious how to extend this solution to the general problem for arbitrary k, without increasing the space bound.

In Lemma 5, we will map this problem to a 2-dimensional problem in computational geometry that can be solved by Chazelle's hive graph data structure [6]. Given n horizontal line segments, the hive graph allows efficient intersection searching along vertical rays. Finding the first horizontal line intersecting a vertical ray requires an orthogonal planar point location query; however, subsequent intersections can be found in sorted order in constant time each. The hive graph requires $O(n)$ space.

Lemma 5. *There exists a data structure for distinct element searching that requires $O(n)$ space and $O(k)$ query time.*

Proof. Let L_i be the set of indices in A that are associated with the leftmost occurrence of an element on or after index i. We can find the leftmost occurrences of the k leftmost distinct elements on or after index i by iterating through L_i in sorted order. However, $\sum_{i=0}^{n-1} |L_i|$ can be $\Omega(n^2)$ so we cannot afford to explicitly store all these sets.

Consider an index ℓ. Clearly, $\ell \in L_\ell$ and $\ell \notin L_i$ for $i > \ell$. Consider the first occurrence of $A[\ell]$ to the left of index ℓ at index ℓ', if it exists. Then $\ell \notin L_i$ for $i \leq \ell'$. However, for $\ell' < i \leq \ell$, $\ell \in L_i$. We associate ℓ with a horizontal segment with x-interval $(\ell', \ell]$ and with y-value ℓ. If no such index ℓ' exists, then we associate ℓ with a horizontal segment with x-interval $(-\infty, \ell]$ and with y-value ℓ. We thus have n horizontal segments. We build Chazelle's hive graph data structure [6] on these segments.

By the construction of the x-intervals of our segments, a segment intersects the vertical line $y = i$ if and only if it is associated with an index ℓ such that

$\ell \in L_i$. Since the y-value of a segment associated with ℓ is ℓ, the segments are sorted along the vertical line in the order of their associated indices. Thus, to find the k leftmost indices in L_i, we query the hive graph for the horizontal segments with a vertical ray from $(i, 0)$ to (i, ∞). The cost of Chazelle's query algorithm is $O(t_{\mathrm{PL}}(n)+k)$ time, where $t_{\mathrm{PL}}(n)$ denotes the cost of a point location query in an orthogonal subdivision of size $O(n)$. The overall query time would then be $O(\log\log n + k)$ if we use the best known linear-space data structure for orthogonal point location of Chan [4].

To reduce the query time to $O(k)$, our key idea is to observe that there are only n distinct vertical rays with which we query the hive graph, and hence only n distinct points with which we do point location. Thus, we can perform the orthogonal point location component of each query during preprocessing and store each resulting node in the hive graph in a total of $O(n)$ space. (In fact, since all the query rays originate from points on the x-axis, the batched point locations are one-dimensional and can be handled easily in our application.) □

Corollary 1. *There exists a data structure for the α-minority range query problem that requires $O(n)$ space and $O(1/\alpha)$ query time.*

Proof. By Lemmas 4 and 5. □

4 Range Majority

We now consider the α-majority range query problem. Recently, Gagie et al. [11] describe an $O(n \log n)$-space data structure that supports α-majority in $O(1/\alpha)$ time, where α is specified at query time. In this section we describe a different α-majority range query data structure whose asymptotic space and time costs match those of Gagie et al. Previous work by Durocher et al. [9] considers the β-majority range query problem, where β is specified during preprocessing; their data structure requires $O(n \log(1/\beta + 1))$ space and supports queries in $O(1/\beta)$ time.

We begin by noting that a β-majority data structure can be adapted to support α-majority at the cost of increased space. Consider $\log n$ instances of the β-majority data structure of Durocher et al. [9], each with respective values $\beta = 2^{-i}$, for $i = 1, \ldots, \log n$, for a total of $O(n \log^2 n)$ space. For any query with parameter α, there is a data structure for which $1/\alpha \leq 1/\beta$ but $1/\beta \in O(1/\alpha)$. Querying this data structure results in a superset of the α-majorities of size $O(1/\alpha)$. The data structure, having counted the frequencies of each of these elements, can then filter out the α-minorities in $O(1/\alpha)$ time. Our effort now turns to solving the problem in $O(n \log n)$ space and $O(1/\alpha)$ query time using an entirely different approach.

Next we consider a related problem: reporting the top k most frequent elements in a query range where k is specified at query time. We call this problem the *top-k range query problem* while warning the reader not to confuse it with reporting the top k highest valued elements. We use a variation on the technique of Lemma 5 in order to support one-sided queries in $O(n)$ space and $O(k)$ query time. We note that the resulting data structure is a persistent version of Lemma 3 in which all updates are provided offline.

Lemma 6. *There exists a data structure for the one-sided top-k range query problem that requires $O(n)$ space and $O(k)$ query time.*

Proof. Assume our one-sided queries take the form $A[0 : i]$ for $0 \leq i \leq n-1$. Consider the frequencies of the elements as we enlarge the one-sided range from left to right. Say an element has frequency f for ranges $A[0 : i]$ through $A[0 : j]$ and this range of ranges is maximal. We construct a horizontal segment with x-interval $[i, j+1)$ and with y-value f. We repeat for all elements and for all $f > 0$ and arbitrarily perturb the y-values for any segments that overlap.

In total, we construct $\Delta \leq n$ segments with y-value 0: one segment corresponding to each distinct element having frequency 0 in a vacuous subarray. Each element of A causes a single change in frequency of a single element, which results in one additional segment. So, in total we construct $O(n)$ segments. We build Chazelle's hive graph data structure [6] on these segments.

For every distinct element e in $A[0 : i]$ there is a horizontal segment with x-interval $[\ell, r+1)$ intersecting the vertical line $y = i$ with $A[\ell] = e$ and $\text{freq}_{A[0:i]}(e) = f$. These horizontal segments are sorted along the vertical line in the order of frequency. To find the k most frequent elements in $A[0 : i]$, we query the hive graph for the first k horizontal segments intersecting the vertical ray from (i, n) to $(i, -\infty)$. As in Lemma 5, there are only n distinct queries to the hive graph, so we can perform the orthogonal point location component of each query during preprocessing at a cost of $O(n)$ space to store the resulting nodes of the hive graph. For each segment that the hive graph reports, we report $A[\ell]$ where ℓ is the left x-coordinate of the segment. □

Observe also that the index of the leftmost endpoint of the horizontal segment associated with a reported element is the index of the rightmost occurrence of the element in $A[0 : i]$. Top-k queries are not decomposable in the sense that, given a partition of a range R into two subranges R_1 and R_2, there is no relationship between the top k most frequent elements in R_1, R_2, and R. As observed by Karpinski and Nekrich [15], given the same partition of R, an α-majority in R must either be an α-majority in R_1 or R_2. Since α-majority queries are decomposable in this way, and since all α-majorities are amongst the top $1/\alpha$ most frequent elements, we can now apply the range tree to support two-sided α-majority queries.

Theorem 3. *There exists a data structure for the α-majority range query problem that requires $O(n \log n)$ space and $O(1/\alpha)$ query time.*

Proof. We build the data structure of Lemma 6 on array A. We divide A into two halves and recurse in both halves to create a range tree. The total space consumption of all top-k data structures is thus $O(n \log n)$. We also include a data structure for lowest common ancestor queries in the range tree. We use this data structure to decompose a two-sided query into one-sided queries in two nodes of the range tree. There are succinct data structures for LCA that require only $O(n)$ bits of space and $O(1)$ time (e.g., [19]). We also build the arrays required to support the queries of Lemma 1.

We decompose a two-sided query into one-sided queries in two nodes of the range tree in $O(1)$ time. For each one-sided query we find the $1/\alpha$ most frequent elements using the top-k data structures in $O(1/\alpha)$ time. By the decomposability of α-majority queries as observed by Karpinski and Nekrich [15], our $O(1/\alpha)$ most frequent elements in both one-sided ranges are a superset of the α-majorities of the original two-sided query. Since the top-k data structures report for each element occurrences that are closest to one of the boundaries of the two-sided range, we can apply Lemma 1 to check which of the $O(1/\alpha)$ most frequent elements are in fact α-majorities in constant time each. □

5 Discussion

Using binary rank and select data structures and bit packing, Chan et al. [5] reduce the range mode query time from $O(\sqrt{n})$ to $O(\sqrt{n/\log n})$ without increasing the data structure's space beyond $O(n)$. Unlike the frequency of the mode, the frequency of the least frequent element does not vary monotonically over a sequence of elements. Furthermore, unlike the mode, when the least frequent element changes, the new element of minimum frequency is not necessarily located in the block in which the change occurs. Consequently, the techniques of Chan et al. do not seem immediately applicable to the least frequent range query problem; it remains open whether $o(\sqrt{n})$ query time is possible in $O(n)$ space.

We have described a data structure for the range least frequent element problem achieving $O(\sqrt{n})$ query time with $O(n^{3/2})$ preprocessing time, and given a lower bound by reduction from boolean matrix multiplication under which least frequent element with $o(n^{1/2-\epsilon})$ query time and $o(n^{3/2-\epsilon})$ preprocessing time would imply matrix multiplication in $o(n^{3-2\epsilon})$ time by purely combinatorial means. We have also given a data structure achieving $O(1/\alpha)$ query time in $O(n)$ space on the range α-minority problem; and one achieving $O(1/\alpha)$ query time in $O(n \log n)$ space on the range α-majority problem, matching the bounds achieved by that of Gagie et al. [11]. The greater space required by current α-majority data structures compared to that required by current α-minority data structures suggests that further improvement may be possible; whether α-majority range queries can be supported in $o(n \log n)$ space and $O(1/\alpha)$ query time remains open.

Acknowledgements. The authors thank Patrick Nicholson for insightful discussion of the α-majority range query problem.

References

1. Bender, M.A., Farach-Colton, M.: The LCA Problem Revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)

2. Bose, P., An, H.-C., Morin, P., Tang, Y.: Approximate Range Mode and Range Median Queries. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 377–388. Springer, Heidelberg (2005)
3. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. Theor. Comp. Sci. 412(24), 2588–2601 (2011)
4. Chan, T.M.: Persistent predecessor search and orthogonal point location on the word RAM. In: Proc. ACM-SIAM SODA, pp. 1131–1145 (2011)
5. Chan, T.M., Durocher, S., Larsen, K.G., Morrison, J., Wilkinson, B.T.: Linear-space data structures for range mode query in arrays. In: Proc. STACS, vol. 14, pp. 291–301 (2012)
6. Chazelle, B.: Filtering search: A new approach to query-answering. SIAM J. Comp. 15(3), 703–724 (1986)
7. Demaine, E.D., Landau, G.M., Weimann, O.: On Cartesian Trees and Range Minimum Queries. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 341–353. Springer, Heidelberg (2009)
8. Durocher, S.: A simple linear-space data structure for constant-time range minimum query. CoRR, abs/1109.4460 (2011)
9. Durocher, S., He, M., Munro, J.I., Nicholson, P.K., Skala, M.: Range Majority in Constant Time and Linear Space. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 244–255. Springer, Heidelberg (2011)
10. Elmasry, A., He, M., Munro, J.I., Nicholson, P.K.: Dynamic Range Majority Data Structures. In: Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 150–159. Springer, Heidelberg (2011)
11. Gagie, T., He, M., Munro, J.I., Nicholson, P.K.: Finding Frequent Elements in Compressed 2D Arrays and Strings. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 295–300. Springer, Heidelberg (2011)
12. Gagie, T., Puglisi, S.J., Turpin, A.: Range Quantile Queries: Another Virtue of Wavelet Trees. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 1–6. Springer, Heidelberg (2009)
13. Greve, M., Jørgensen, A.G., Larsen, K.D., Truelsen, J.: Cell Probe Lower Bounds and Approximations for Range Mode. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 605–616. Springer, Heidelberg (2010)
14. Jørgensen, A.G., Larsen, K.D.: Range selection and median: Tight cell probe lower bounds and adaptive data structures. In: Proc. ACM-SIAM SODA, pp. 805–813 (2011)
15. Karpinski, M., Nekrich, Y.: Searching for frequent colors in rectangles. In: Proc. CCCG, pp. 11–14 (2008)
16. Krizanc, D., Morin, P., Smid, M.: Range mode and range median queries on lists and trees. Nordic J. Computing 12, 1–17 (2005)
17. Petersen, H.: Improved Bounds for Range Mode and Range Median Queries. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 418–423. Springer, Heidelberg (2008)
18. Petersen, H., Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space. Inf. Proc. Let. 109, 225–228 (2009)
19. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: Proc. ACM-SIAM SODA, pp. 134–149 (2010)

Asynchronous Rumor Spreading in Preferential Attachment Graphs

Benjamin Doerr[1], Mahmoud Fouz[2], and Tobias Friedrich[1,2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Universität des Saarlandes, Saarbrücken, Germany

Abstract. We show that the asynchronous push-pull protocol spreads rumors in preferential attachment graphs (as defined by Barabási and Albert) in time $O(\sqrt{\log n})$ to all but a lower order fraction of the nodes with high probability. This is significantly faster than what synchronized protocols can achieve; an obvious lower bound for these is the average distance, which is known to be $\Theta(\log n/\log\log n)$.

1 Introduction

Online social networks like *Facebook* and *Twitter* are changing the way people communicate, organize and act collectively. They are starting to take the lead over traditional news media in their ability to spread news at a remarkable speed. One striking example was the first picture of US Airways Flight 1549's crash landing on the Hudson River, which became known to a broad audience through Twitter even before TV channels started to report on the accident.

The theoretical model most widely used for social networks is the so-called *preferential attachment* (PA) model, which was introduced in a seminal paper by Barabási and Albert [1]. It builds on the paradigm that new vertices attach to already present vertices with a probability proportional to their degree. Several papersprove that this model indeed enjoys many properties observed in social networks and many other real world networks, e.g., a power law distribution of the vertex degrees, a small diameter and a small average degree [2, 4]. The precise definition of the PA model can be found in Section 2. Note that later extended definitions for PA graphs were given (with the preference not anymore proportional to the degree); in this paper, we shall always refer to the original one.

In this paper, we revisit the *rumor spreading* problem in PA graphs, i.e., the spread of one piece of information in a graph. The classical rumor spreading process is modeled on a discrete time line. A simple protocol assumes that in each time step (or *round*) every node that knows the rumor forwards it to a randomly chosen neighbor. This is known as the *push strategy.* For many network topologies, this strategy is a very efficient way to spread a rumor. Let n denote the number of vertices of a graph. Then the push model with high probability (i.e., with probability $1 - o(1)$) sends the rumor to all vertices in time $\Theta(\log n)$, if the graph is a complete graph [19], a hypercube [15], an Erdős-Rényi random

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 307–315, 2012.

graph $G_{n,p}$ with $p \geq (1+\varepsilon)\log(n)/n$ [15], or a random regular graph [17]. In contrast to this, Chierichetti, Lattanzi, and Panconesi [7] showed that the push model with non-vanishing probability needs $\Omega(n^{\alpha})$ rounds on PA graphs for some $\alpha > 0$.

Opposite to the push strategy is the *pull strategy*: each vertex in each round contacts a random neighbor and learns the rumor if its contact knows the rumor. There is a symmetry between the two models [6, 10], hence these results also hold for the pull model.

Karp, Schindelhauer, Shenker, and Vöcking [22] pointed out that for complete graphs, the pull strategy is inferior to the push strategy until roughly $n/2$ vertices are informed, and then the pull strategy becomes more effective. This motivates to combine both approaches. In this so-called *push-pull strategy* each vertex contacts another vertex chosen uniformly at random among its neighbors. It *pushes* the rumor in case it has the rumor, and *pulls* the rumor in case the neighbor has the rumor. For complete graphs and many Erdős-Rényi random graphs, this protocol also needs $\Theta(\log n)$ rounds, though with better implicit constants [9, 13, 22]. Its main advantage here is that it allows to define protolls using fewer messages. Chierichetti et al. [6] relate the broadcast time of the push-pull strategy to the conductance of graphs; graphs with conductance Φ have a broadcast time of $O\big(\log^2(\Phi^{-1})\,\Phi^{-1}\,\log n\big)$ with high probability. Giakkoupis [20] recently improved this bound to $O(\Phi^{-1}\log n)$ which is tight.

For preferential attachment graphs, however, the push-pull strategy is much better than push or pull alone. Chierichetti et al. [7] showed that with this strategy, $O(\log^2 n)$ rounds suffice with high probability. Recently, we showed that in fact the push-pull strategy succeeds to inform all nodes in $\Theta(\log n)$ rounds [11]. Surprisingly, if the push-pull strategy is slightly modified to prevent that a node contacts the same neighbor twice in a row, then with high probability already $\Theta(\log n/\log\log n)$ rounds suffice [11], which is the diameter of the PA graph.

All these results assume a *synchronized* model, in which all nodes take action simultaneously at discrete time steps. In many applications and certainly in real-world social networks, this assumption is not very plausible. One can also argue (see, e.g., [5]) that time-synchronization contradicts the idea of a self-organized broadcasting protocol. Boyd et al. [5] therefore proposed an *asynchronous time model* with a *continuous* time line. Each node has its own clock that ticks at the times of a rate 1 Poisson process independent from the clocks of other nodes. The protocol now specifies for every node what to do when its own clock ticks.

The rumor spreading problem in the asynchronous time model has so far received less attention. The push-pull protocol in this model, however, turns out to be closely related to Richardson's model for the spread of a disease and to first-passage percolation. In this sense, for the hypercube, Fill and Pemantle [16] and Bollobás and Thomason [3] showed that the asynchronous push-pull protocol spreads a rumor to all nodes in time $\Theta(\log n)$. Similarly, for the complete graph, Janson [21] showed a bound of $\Theta(\log n)$. Note that these bounds match the same asymptotics as in the synchronized case. We also suspect that the same bounds hold in case all but $o(n)$ nodes are to be informed.

Fountoulakis, Panagiotou, and Sauerwald [18] have recently studied the push-pull protocol in the asynchronous time model for random graphs in the Ching-Lu model [8] with a given expected degree distribution that follows a power law with exponent in $(2, 3)$. For these graphs, they show a constant runtime to inform $n - o(n)$ nodes. Note that these graphs are quite different from our PA graphs, e.g., their average diameter is known to be $\Theta(\log \log n)$ (see [8]), whereas for PA graphs the average diameter is also $\Theta(\log n/ \log \log n)$ (see [12]).

Our Results: We study the push-pull protocol in the asynchronous time model on PA graphs and prove that it spreads a rumor in time $O(\sqrt{\log n})$ to $n - o(n)$ nodes in the PA model with high probability. The protocol thus beats the average distance of $\Theta(\log n/ \log \log n)$, which is a natural lower bound for the synchronized protocol achieving this aim. To inform all nodes, however, our protocol is shown to need $\Theta(\log n)$ time. This is mainly due to few nodes that require $\Omega(\log n)$ time to contact or be contacted by a neighbor for the first time.

These results show that the asynchronous push-pull protocol behaves quite differently than the synchronized one, despite the fact that each node still contacts one neighbor per time unit on average. The discrepancy between informing all nodes and almost all nodes reflects an often observed 'long tail' behavior in real world networks. Such effects are less visible in the synchronized case [11].

2 Precise Model and Preliminaries

Preferential attachment graphs were first introduced by Barabási and Albert [1]. In this work, we follow the formal definition of Bollobás et al. [2, 4]. Let G be an undirected graph. We denote by $\deg_G(v)$ the degree of a vertex v in G.

Definition 1 (Preferential attachment graph). *Let $m \geq 2$ be a fixed parameter. The random graph G_m^n is an undirected graph on the vertex set $V := \{1, \ldots, n\}$ inductively defined as follows.*

G_m^1 consists of a single vertex with m self-loops. For all $n > 1$, G_m^n is built from G_m^{n-1} by adding the new node n together with m edges $e_n^1 = \{n, v_1\}, \ldots, e_n^m = \{n, v_m\}$ inserted one after the other in this order. Let $G_{m,i-1}^n$ denote the graph right before the edge e_n^i is added. Let $M_i = \sum_{v \in V} \deg_{G_{m,i-1}^n}(v)$ be the sum of the degrees of all the nodes in $G_{m,i-1}^n$. The endpoint v_i is selected randomly such that $v_i = u$ with probability $\deg_{G_{m,i-1}^n}(u)/(M_i + 1)$, except for n that is selected with probability $(\deg_{G_{m,i-1}^n}(n) + 1)/(M_i + 1)$.

This definition implies that when e_n^i is inserted, the vertex v_i is chosen with probability proportional to its degree (except for $v_i = n$). Since many real-world social networks are conjectured to evolve using similar principles, the PA model can serve as a model for social networks. Another property observed in many real-world networks has been formally proven for preferential attachment graphs, namely that the degree distribution follows a power-law [4].

For $m = 1$ the graph is disconnected with high probability; so we focus on the case $m \geq 2$. Under this assumption, Bollobás and Riordan [2] showed that the diameter is only $\Theta(\log(n)/\log\log n)$ with high probability.

With a slight abuse of notation we write $(u, v) \in E$ or $(v, u) \in E$ both to denote $\{u, v\} \in E$. The definition of G_m^n can lead to multiple edges and self-loops, though they typically make up only a vanishing fraction of the edges.

We examine the following broadcasting protocol.

Definition 2 (Asynchronous push-pull strategy). *Each node has a clock that ticks at the times of a rate 1 Poisson process. Whenever the clock of a vertex u ticks, it chooses uniformly at random a neighbor v. If u knows the rumor, it sends the rumor to v ("push"). If v knows the rumor, it sends the rumor to u ("pull").*

We say that an edge (u, v) *fires*, whenever the clock of node u ticks and u contacts v. We call the time span between two ticks of a clock a *round*. The length of a round is *exponentially* distributed with mean 1.Since the exponential distribution is memoryless, the length of a round is independent over time. The following elementary lemma shows that also the time when a node contacts a *specific* neighbor is exponentially distributed.

Lemma 1. *Let u be a node of degree d that is connected to a node v. Let T denote the time span until u contacts v. Then, $\mathbb{P}[T > x] = e^{-x/d}$.*

3 Statement of Results

Theorem 1. *With probability $1 - o(1)$, the asynchronous push-pull protocol broadcasts a rumor from any node of G_m^n to (i) all but $o(n)$ nodes in time $O(\sqrt{\log n})$, (ii) and to all nodes in time $\Theta(\log n)$.*

The proofs of the upper bounds in Theorem 1 consist of three main steps. In Section 4.3, we analyze the time needed until the rumor reaches a so-called *useful node*. Roughly speaking, a node is useful if its degree is at least polylogarithmic (see Section 4.2 for details). We prove that a useful node is reached in time $O((\log\log n)^2)$ with probability $1 - o(1)$ and in time $O(\log n)$ with probability $1-o(n^{-2})$. The later bound is used for the case when all nodes are to be informed.

The core of the proof (see Section 4.4) consists of showing that once a useful node u has been informed, within $O(\sqrt{\log n})$ time the rumor is propagated to node 1. To this aim, we show that there is a short path from u to 1 such that every second node has degree exactly m that is traversed in time $O(\sqrt{\log n})$. To prove such a fast traversal we exploit edges that fire fast. In particular, we use the fact that the minimum of k i.i.d. exponential random variables with mean 1 is also exponentially distributed with mean $1/k$.

The result then follows from the following symmetry property.

Lemma 2. *Assume that if the rumor starts in node u, it reaches node v in time t with probability p. This implies the reverse statement: if the rumor is initiated by v, then it reaches u in time t with probability p.*

4 Analysis of the Asynchronous Push-Pull Model

4.1 Alternative Model

In the random process generating G_m^n the random decisions made at each step depend heavily on the previous random decisions. Bollobás and Riordan [2] therefore suggested an alternative way of generating G_m^n that is more accessible. We first describe the model for $m = 1$ and then generalize it to arbitrary m.

Let (x_i, y_i) for $i \in [n] := \{1, 2, \dots, n\}$ be n independently and uniformly chosen pairs from $[0,1] \times [0,1]$. With probability one, all these numbers are distinct. By reordering each pair if necessary, we assume that $x_i < y_i$ for every $i \in [n]$. Suppose that after relabeling, $y_1 < y_2 < \cdots < y_n$. We set $W_0 := 0$ and $W_i := y_i$ for $i \in [n]$. The graph G_1^n is now defined by having an edge (i, j) if and only if $W_{j-1} < x_i < W_j$. Define $w_j := W_j - W_{j-1}$.

Similarly, for G_m^n, we sample mn pairs $(x_{i,j}, y_{i,j})$ independently and uniformly from $[0,1] \times [0,1]$ with $x_{i,j} < y_{i,j}$ for $i \in [n]$ and $j \in [m]$. We relabel the variables such that $y_{i,j}$ is increasing in lexicographic order: $y_{1,1} < y_{1,2} < \cdots < y_{1,m} < y_{2,1} < \cdots < y_{n,1} < \cdots < y_{n,m}$. We set $W_0 := 0$ and $W_i := y_{i,m}$ for $i \in [n]$. The graph is now defined by having an edge (i, j) for each $k \in [m]$ such that $W_{j-1} < x_{i,k} < W_j$. As before, define $w_j = W_j - W_{j-1}$. We write $\ell_{i,k}$ for the node j such that $W_{j-1} < x_{i,k} < W_j$. Note that given $y_{1,1}, \dots, y_{n,m}$, the random variables $x_{1,1}, \dots, x_{n,m}$ are independent with x_{i_k} being chosen uniformly from $[0, y_{i,k}]$. We instead assume that if $y_{1,1}, \dots, y_{n,m}$ are given, then each $x_{i,k}$ is chosen independently and uniformly from $[0, W_i]$. By this slight modification, we can work with the values of the W_i's and ignore the values of the $y_{i,j}$'s. This modification only increases the probability of a loop at i. It is straightforward to check that each step of our proof remains valid if the probability of a loop is not increased. Thus, the validity of our proof is not affected.

We give a few properties of the alternative model, that are useful in the analysis. Let $s = 2^a$ be the smallest power of 2 larger than $\log^{10} n$, and let 2^b be the largest power of 2 smaller than $2n/3$. Let $I_t = [2^t + 1, 2^{t+1}]$.

Lemma 3 (Bollobás and Riordan [2]). *Let $m \geq 2$ be fixed. Using the definitions above, each of the following five events holds with probability $1 - o(1)$.*

- $E_1 := \left\{|W_i - \sqrt{i/n}| \leq \frac{1}{10}\sqrt{i/n} \text{ for all } i \in [s, n]\right\}$
- $E_2 := \left\{|\{i \in I_t \mid w_i \geq 1/(10\sqrt{in})\}| \geq 2^{t-1} \text{ for all } t \in [a, b)\right\}$
- $E_3 := \{w_1 \geq \frac{4}{\log(n)\sqrt{n}}\}$
- $E_4 := \{w_i \geq \log^2(n)/n \text{ for all } i < n^{1/5}\}$
- $E_5 := \{w_i < \log^2(n)/n \text{ for all } i \geq n/2\}$.

Note that the event E_5 is slightly adjusted for our purposes. In the original paper, the authors show that for $i \geq n/\log^5 n$; we have $w_i < n^{-4/5}$. It is easy to check that (essentially) the same proof holds for the above version.

Instead of working directly with the alternative model where the W_i's are random variables, we use the following *typical social network* model where the

W_i's are fixed numbers that satisfy the properties $E_1, \dots, E_5$. Since by Lemma 3, these properties hold with high probability, all results proven for a typical social network model carry over to G_m^n with high probability. Let $0 < W_1 < \dots < W_n < 1$ be distinct real numbers and let $w_i = W_i - W_{i-1}$. Assume that $W_1, \dots, W_n$ satisfy the properties $E_1, \dots, E_5$. A typical social network $G_m(W_1, \dots, W_n)$ is obtained by connecting each node i with the nodes $\ell_{i,1}, \dots, \ell_{i,m}$, where each $\ell_{i,k}$ is a node chosen randomly with $\mathbb{P}[\ell_{i,k} = j] = w_j / W_i$ for all $j \le i$.

We always assume to have a typical social network $G := G_m(W_1, \dots, W_n)$.

4.2 Useful Nodes

We use the notion of a *useful* node that was introduced by Bollobás and Riordan [2]. A node i is useful if $w_i \ge \log^2(n)/n$. Note that we are slightly relaxing the original definition in [2] where the authors also assumed that $i \le n/\log^5(n)$. We have by E_5 that $i < n/2$ for all useful nodes. Furthermore by E_4, every $i < n^{1/5}$ is useful. The following properties of non-useful nodes were proved in [11].

Lemma 4. *With probability $1 - o(1)$, the following event holds*

- $E_6 := \{\deg_G(v) \le 5m \log^2 n$ *for all non-useful* $v\}$.

Lemma 5. *Assume that E_6 holds. With prob. $1 - n^{-1/5+o(1)}$, we have*

- $E_7 := \{$*for all non-useful v, there exists at most one cycle whose nodes are all connected to v via non-useful paths of length at most* $\frac{\log n}{(\log \log n)^2}\}$.

Lemma 6 (Bollobás and Riordan [2]). *Let v be a fixed non-useful node. Then for all $k \in [m]$, the prob. that $\ell_{v,k}$ is a useful node is at least $\log^{-3} n$. This event is independent from all other random decisions $\ell_{v',k'}$ with $(v', k') \neq (v, k)$.*

Note that in the original lemma, the authors only state a bound on the probability that $\ell_{v,1}$ is a useful node. However, the same proof yields the above version. Also, Lemma 6 remains valid if we condition on E_6.

4.3 Informing the First Useful Node

Let $G = G_m(W_1, \dots, W_n)$ be a typical social network. Assume that also E_6 and E_7 hold. In this section, all probabilities are taken over the product space of the random graph G and the random decisions of the rumor spreading process.

Lemma 7. *Let u be a fixed node. The rumor initiated by u reaches a useful node in time $O((\log \log n)^2)$ with probability $1 - o(1)$, and in time $O(\log n)$ with probability $1 - o(n^{-2})$.*

4.4 Informing Node 1

Similar to the synchronized case, we use constant degree nodes to establish *fast links* between large degree nodes. More precisely, once a neighbor of a constant degree node is informed, the time until it has pulled the rumor from this neighbor and pushed it to one specific neighbor is (essentially) exponentially distributed. Thus, independent of their own degrees, two nodes that are connected via a third node of constant degree exchange information in time exponentially distributed.

Starting from one informed useful node, we study how fast the rumor spreads to the surrounding 'neighborhoods' of nodes. We consider neighborhoods alternating between small nodes and *good* nodes i of relatively large weight w_i. The small nodes act as fast links between the levels of good nodes that ensure a large expansion. In particular, we make use of the fact that a good node i has a high degree and since every small neighbor of i independently pulls the rumor in time exponentially distributed, we can argue that a considerable fraction of the small neighbors of i will be informed very fast. The more neighbors of informed nodes there are, the faster the rumor will spread to sufficiently many neighbors that form the next level of informed node. In contrast, in the synchronized case, it would always take at least one time step for a neighbor to pull the rumor.

We consider informed neighborhoods at suitably chosen time steps on the continuous time line. The smaller these steps are chosen, the smaller the achieved expansion factor is at each step. On the other hand, smaller time steps allow us to progress faster through the different neighborhood levels. By carefully choosing each step size, we can balance out these opposing effects in order to achieve the following runtime.

Theorem 2. *Let $W_1, \ldots, W_n$ be such that $E_1, \ldots, E_5$ are satisfied. Let G be a random graph from $G_m(W_1, \ldots, W_n)$. Let $v \in [n]$ be a useful node. With probability $1 - o(n^{-1})$, using the asynchronous push-pull protocol, a rumor present at v reaches node 1 in $O(\sqrt{\log n})$ steps.*

For our argument using fast links, we will need many nodes of constant degree. The following simple lemma proves that there is a linear number of nodes $i \in \left[\frac{2}{3}n, n\right]$ that have a degree equal to m. We call such nodes *small.* If not explicitly stated, all probabilities in this section are taken over the random graph $G_m(W_1, \ldots, W_n)$, where $W_1, \ldots, W_n$ are given numbers that satisfy properties $E_1, \ldots, E_5$.

Lemma 8. *Let $\varepsilon_m := \frac{1}{8}e^{-3m}$. With probability $1 - e^{-\Omega(n)}$, there are at least $\varepsilon_m n$ small nodes in $\left[\frac{2}{3}n, n\right]$.*

Crucial for a large expansion in each step are *good* nodes of large weight. We say a node i is *good* if

$$i \in [s+1, 2^b] \text{ and } w_i \geq 1/(10\sqrt{in}), \tag{1}$$

where, as before, $s = 2^a$ is the smallest power of 2 larger than $\log^{10} n$ and 2^b is the largest power of 2 smaller than $\frac{2}{3}n$. Let u be a useful node. Let $t_0 < t_0' < t_1 < t_1' < \ldots$ denote discrete time steps to be specified later. We consider neighborhoods of u that are informed in these time intervals. In particular, we define sets Γ_k and Γ_k' recursively as follows. We set $\Gamma_0 = \{u\}$. Given the set Γ_k, Γ_k' consists of all *small* nodes $i \geq \frac{2}{3}n$ that contact a neighbor in Γ_k in time

$[t_k, t'_k]$ and have not been included in any Γ'_ℓ with $\ell \leq k-1$. Similarly, Γ_k is defined as the set of all *good* nodes that are contacted by a neighbor in Γ'_{k-1} in time $[t'_{k-1}, t_k]$ and have not been included in any Γ_ℓ with $\ell \leq k-1$. Note that for all $k \geq 0$, Γ_k only contains nodes $i < \frac{2}{3}n$, while Γ'_k only contains nodes $i \geq \frac{2}{3}n$. This is true for Γ_0 since u is useful and by E_5, all useful nodes are smaller than $n/2$. We define the *weight* of a set Γ_k by

$$f_k := \begin{cases} w_u & \text{if } k = 0 \\ \sum_{i \in \Gamma_k} \frac{1}{\sqrt{in}} & \text{if } k \geq 1. \end{cases} \tag{2}$$

Since for $k \geq 1$, Γ_k only contains good nodes, and by definition, $w_u = f_0$, we have for $k \geq 0$,

$$\sum_{i \in \Gamma_k} w_i \geq f_k/10. \tag{3}$$

We denote by $N_k = \Gamma_0 \cup \Gamma_1 \cup \cdots \cup \Gamma_k$ (note that the Γ'_i are not included). Let $C_0 \subseteq \left[\frac{2}{3}n, n\right]$ be the set of small nodes and for $k \geq 1$, let $C_k = C_0 \setminus \{\Gamma'_0, \ldots, \Gamma'_{k-1}\}$ be the set of small nodes excluding nodes in $\Gamma'_0, \Gamma'_1, \ldots, \Gamma'_{k-1}$. By Lemma 8, we have $C_0 \geq \varepsilon_m n$ with probability $1 - e^{-\Omega(n)}$.

The next lemma shows that we achieve an exponential expansion in terms of f_k in each level as long as there is still a linear number of small nodes in C_k and similarly, as long as for each interval $I_t := [2^t + 1, 2^{t+1}]$, where $t \in [a, b)$, there are still 2^{t-2} good nodes that are not N_k.

Lemma 9. *Let $c > 0$ be a sufficiently large constant, $k \geq 0$ be such that $\log^4(n)/\sqrt{n} \geq f_k \geq \log^2(n)/n$ and $|C_k| \geq \varepsilon_m n/2$. Let $\Delta_k = c \log_2^{-1/2}(\varepsilon_m f_k n/\log^2 n)$. Set $t'_k := t_k + \Delta_k$ and $t_{k+1} := t'_k + \Delta_k$. Then given C_k and $\Gamma_0, \Gamma'_0, \Gamma_1, \Gamma'_1, \ldots, \Gamma_k$, with prob. $1 - O(n^{-6/5})$, one of the following is satisfied. (i) $|N_{k+1} \cap I_t| \geq 2^{t-2}$, for some $t \in [a, b)$, or (ii) $f_{k+1} \geq 2f_k$.*

5 Conclusion

We have shown that for PA graphs the asynchronous push-pull protocol informs almost all nodes in $O(\sqrt{\log n})$ time. This shows, in an even stronger way than the previous $\tilde{\Theta}(\log n)$ bounds for the synchronized protocol [11], that randomized rumor spreading is very effective in network topologies resembling real-world networks.

From a broader perspective, our result also indicates that in naturally asynchronous settings, it might be a misleading oversimplification to assume a synchronized protocol.

References

[1] Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science 286, 509–512 (1999)

[2] Bollobás, B., Riordan, O.: The diameter of a scale-free random graph. Combinatorica 24, 5–34 (2004)

[3] Bollobás, B., Thomason, A.: On Richardson's Model on the Hypercube. Cambridge University Press (1997)

[4] Bollobás, B., Riordan, O., Spencer, J., Tusnády, G.: The degree sequence of a scale-free random graph process. Random Structures & Algorithms 18, 279–290 (2001)

[5] Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. IEEE Transactions on Information Theory 52, 2508–2530 (2006)

[6] Chierichetti, F., Lattanzi, S., Panconesi, A.: Almost tight bounds for rumour spreading with conductance. In: 42nd ACM Symposium on Theory of Computing (STOC), pp. 399–408 (2010)

[7] Chierichetti, F., Lattanzi, S., Panconesi, A.: Rumor spreading in social networks. Theoretical Computer Science 412, 2602–2610 (2011)

[8] Chung, F.R.K., Lu, L.: The average distance in a random graph with given expected degrees. Internet Mathematics 1, 91–113 (2003)

[9] Demers, A.J., Greene, D.H., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H.E., Swinehart, D.C., Terry, D.B.: Epidemic algorithms for replicated database maintenance. Operating Systems Review 22, 8–32 (1988)

[10] Doerr, B., Friedrich, T., Sauerwald, T.: Quasirandom Rumor Spreading: Expanders, Push vs. Pull, and Robustness. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 366–377. Springer, Heidelberg (2009)

[11] Doerr, B., Fouz, M., Friedrich, T.: Social networks spread rumors in sublogarithmic time. In: 43rd ACM Symposium on Theory of Computing (STOC), pp. 21–30 (2011)

[12] Dommers, S., van der Hofstad, R., Hooghiemstray, G.: Diameters in preferential attachment models. Journal of Statistical Physics 139, 72–107 (2010)

[13] Elsässer, R.: On the communication complexity of randomized broadcasting in random-like graphs. In: 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 148–157 (2006)

[14] Evans, M., Hastings, N., Peacock, B.: Statistical Distributions, 3rd edn. John Wiley & Sons, Inc. (2000)

[15] Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. Random Structures & Algorithms 1, 447–460 (1990)

[16] Fill, J.A., Pemantle, R.: Percolation, first-passage percolation, and covering times for Richardson's model on the n-cube. Annals of Applied Probability 3, 593–629 (1993)

[17] Fountoulakis, N., Panagiotou, K.: Rumor Spreading on Random Regular Graphs and Expanders. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010, LNCS, vol. 6302, pp. 560–573. Springer, Heidelberg (2010)

[18] Fountoulakis, N., Panagiotou, K., Sauerwald, T.: Ultra-fast rumor spreading in social networks. In: 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1642–1660 (2012)

[19] Frieze, A.M., Grimmett, G.R.: The shortest-path problem for graphs with random arc-lengths. Discrete Applied Mathematics 10, 57–77 (1985)

[20] Giakkoupis, G.: Tight bounds for rumor spreading in graphs of a given conductance. In: 28th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 57–68 (2011)

[21] Janson, S.: One, two and three times $\log n/n$ for paths in a complete graph with random weights. Combinatorics, Probability & Computing 8, 347–361 (1999)

[22] Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumor spreading. In: 41st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 565–574 (2000)

Connectivity Oracles for Planar Graphs

Glencora Borradaile[1,⋆], Seth Pettie[2,⋆⋆], and Christian Wulff-Nilsen[3,⋆⋆⋆]

[1] Oregon State University
glencora@eecs.orst.edu
[2] University of Michigan
pettie@umich.edu
[3] University of Southern Denmark
koolooz@diku.dk

Abstract. We consider dynamic subgraph connectivity problems for planar undirected graphs. In this model there is a fixed underlying planar graph, where each edge and vertex is either "off" (failed) or "on" (recovered). We wish to answer connectivity queries with respect to the "on" subgraph. The model has two natural variants, one in which there are d edge/vertex failures that precede all connectivity queries, and one in which failures/recoveries and queries are intermixed.

We present a d-failure connectivity oracle for planar graphs that processes any d edge/vertex failures in $\mathrm{sort}(d, n)$ time so that connectivity queries can be answered in $\mathrm{pred}(d, n)$ time. (Here sort and pred are the time for integer sorting and integer predecessor search over a subset of $[n]$ of size d.) Our algorithm has two discrete parts. The first is an algorithm tailored to triconnected planar graphs. It makes use of Barnette's theorem, which states that every triconnected planar graph contains a degree-3 spanning tree. The second part is a generic reduction from general (planar) graphs to triconnected (planar) graphs. Our algorithm is, moreover, provably optimal. An implication of Pătraşcu and Thorup's lower bound on predecessor search is that no d-failure connectivity oracle (even on trees) can beat $\mathrm{pred}(d, n)$ query time.

We extend our algorithms to the subgraph connectivity model where edge/vertex failures (but no recoveries) are intermixed with connectivity queries. In triconnected planar graphs each failure and query is handled in $O(\log n)$ time (amortized), whereas in general planar graphs both bounds become $O(\log^2 n)$.

1 Introduction

Algorithms for dynamic graphs have traditionally assumed that the graph evolves according to a completely arbitrary sequence of insertions and deletions of graph elements. This model makes minimal assumptions but often sacrifices efficiency

[⋆] Supported by NSF CCF-0963921.

[⋆⋆] Supported by NSF CAREER grant no. CCF-0746673 and a grant from the US-Israel Binational Science Foundation.

[⋆⋆⋆] Supported by NSERC and MRI.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 316–327, 2012.

for generality. For example, real world networks (router networks, road networks, etc.) *do* change slowly over time. However, the real dynamism of the networks comes from the frequent *failure* of edges/nodes and their subsequent recovery. In this paper we study connectivity problems in the *dynamic subgraph model*, which attempts to accurately model this type of dynamism. It is assumed that there is some fixed underlying graph whose nodes and edges can be *off* (failed) or *on*; queries (connectivity queries, in our case) then answer questions about the subgraph turned on. The power of this model (compared to the fully dynamic graph model) stems from the ability to preprocess the underlying graph in advance.

There are two natural variants of the dynamic subgraph model. In the d-failure version failures and recoveries occur in lockstep: a set F of $d = |F|$ edges/nodes fail together. Our goal is to process F, ideally in $\tilde{O}(d)$ time, in order to answer connectivity queries in $G \backslash F$. Here d may or may not be a parameter of the algorithm. In the fully dynamic model, node/edge failures and recoveries are presented one at a time and intermixed with connectivity queries, whereas in the decremental model the updates are restricted to failures.

Results. We give new algorithms for subgraph connectivity on undirected planar graphs in the d-failure model and the decremental model, all of which require linear preprocessing time. When failures are restricted to edges, we give an especially simple connectivity oracle that processes d edge failures (for any d) in $O(\mathrm{sort}(d, n))$ time and subsequently answers queries in $O(\mathrm{pred}(d, n))$ time. Here sort and pred refer to the time for sorting d integers in the universe $\{1, \ldots, n\}$ and pred for the time for predecessor search, given $\mathrm{sort}(d, n)$ preprocessing time.[1]

The problem becomes more complicated when vertices fail since we cannot, in general, spend time proportional to their degrees. Our second algorithm is a d-failure connectivity oracle for edge and vertex failures with the same parameters (linear preprocessing, $O(\mathrm{sort}(d, n))$ to process d failures, $O(\mathrm{pred}(d, n))$ time per query). It consists of two parts: a solution for triconnected graphs and a generic reduction from d-failure oracles in general graphs to d-failure oracles in triconnected graphs. Triconnectivity plays an important role in the algorithm as it allows us to apply Barnette's theorem [6], which states that every triconnected planar graph contains a degree-3 spanning tree. It is known [27] that predecessor search is reducible to the d-edge/node failure connectivity problem on trees. Our query time is therefore provably optimal. In particular, Pătraşcu and Thorup's lower bound [26] implies that $O(\log\log(n/d))$ query time cannot be beaten in general, even given $O(d\,\mathrm{poly}(\log n))$ time to preprocess the edge/node failures.

Our third algorithm is in the decremental model. In triconnected planar graphs we can support vertex failures in $O(\log n)$ amortized time and connectivity queries in $O(\log n)$ time, whereas in general planar graphs both bounds

[1] It is known that $\mathrm{sort}(d, n) = O(d \log\log d)$ deterministically, $O(d\sqrt{\log\log d})$ randomized [22,23], and $O(d)$ randomized if $d < 2^{\sqrt{\log n} - \epsilon}$ [4]. For predecessor search the bound is $\mathrm{pred}(d, n) = O(\min\{\frac{\log d}{\log\log n}, \frac{\log\log d \cdot \log\log n}{\log\log\log n}\})$ deterministically [20,5] and $O(\log\log(n/d))$ randomized [29,31].

become $O(\log^2 n)$.[2] The logarithmic slowdown comes from a new reduction from (planar) dynamic subgraph connectivity to the same problem in triconnected graphs.

Prior Work on Dynamic Connectivity. Before surveying dynamic subgraph connectivity it is instructive to see what type of “off the shelf” solutions are available using traditional dynamic graph algorithms. The best known dynamic connectivity algorithms for general undirected graphs require $O(\sqrt{n})$ worst case time per edge update [19,17] or $O(\log^2 n)$ time amortized [25]. Vertex updates are simulated with $O(n)$ edge updates. In dynamic planar graphs the best connectivity algorithms take $O(\log n)$ time per edge update [18].

Prior Work on Subgraph Connectivity. The dynamic subgraph model was proposed explicitly by Frigioni and Italiano [21], who proved that in planar graphs, node failures and recoveries could be supported in $O(\log^3 n)$ amortized time per operation and connectivity queries in $O(\log^3 n)$ worst case time. An earlier algorithm of Eppstein et al. [18] implies that edge failures, edge recoveries, and connectivity queries in planar graphs require $O(\log n)$ time. In general graphs, Chan, Pătraşcu, and Roditty [11] (improving [10]) showed that node updates could be supported in amortized time $O(m^{2/3})$, where m is the number of edges, and connectivity queries in $O(m^{1/3})$. Chan et al. [1,10,11] gave numerous applications of subgraph connectivity to geometric connectivity problems. The first algorithm with worst-case guarantees was given by Duan [13], who showed that node updates and queries require only $O(m^{4/5})$ and $O(m^{1/5})$ time, respectively.

In the *d-edge* failure model Pătraşcu and Thorup [27] gave a connectivity oracle for general graphs that processes d failures in $O(d \log^2 n \log\log n)$ time and answers queries in $O(\log\log n)$ time. However, their structure requires *exponential* preprocessing time; a variant constructible in polynomial time has a slower update time: $O(d \log^{5/2} \log\log n)$. Duan and Pettie [15] gave a connectivity oracle in the *d-node* failure model occupying $O(mn^{\epsilon})$ space that processes failures in $O(\text{poly}(d, \log n))$ time and answers queries in $O(d)$ time, where $\text{poly}(d, n)$ depends on ϵ. They also showed that a d-edge failure oracle could be constructed in $\tilde{O}(n)$ time with $O(d^2 \log\log n)$ update time and $O(\log\log n)$ query time.

The *distance sensitivity* oracles avoiding 1 node failure [12,8] or 2 node failures [14] also, as a special case, answer 1- and 2-failure connectivity queries on *directed* graphs in $O(1)$ and $O(\log n)$ time, respectively. These data structures occupy $\tilde{O}(n^2)$ space.

Overview. Section 2 reviews notation and terminology. In Section 3 we describe our planar d-edge failure connectivity oracle. In Section 4 we give a d-vertex failure oracle for triconnected planar graphs and in Section 5 we extend it to a decremental subgraph connectivity oracle for triconnected graphs. The full manuscript [9] contains reductions from general (planar) graphs to triconnected (planar) graphs.

[2] These bounds are amortized over the actual number f of failures, i.e., in triconnected graphs processing f failures takes $O(f \log n)$ time total.

2 Definitions, Notation, and Basic Results

We assume that all graphs considered are undirected. A *planar graph* is a graph that can be drawn in the plane such that no two edges cross. We refer to such a drawing as a *plane graph.* A plane graph $G = (V, E)$ partitions the plane into maximal open connected sets and we refer to the closure of these sets as the *faces* of G. If G is connected, we define a plane graph, called the *dual graph* G^* of G, as follows. Associated with each face f of G is a vertex in G^* which we identify with f and which we draw inside f. For each edge e in G, there is an edge (f_1, f_2) in G^*, where f_1 and f_2 are the faces in G incident to e. We draw (f_1, f_2) such that it crosses e exactly once and crosses no other edge in G or in G^*. We identify (f_1, f_2) with e. It can be shown that G^* is also connected and that $(G^*)^* = G$. In particular, each face in G^* corresponds to a vertex in G. We refer to G as the *primal* graph.

For vertices u and v in a rooted tree T, we let $\text{lca}_T(u, v)$ denote the lowest common ancestor of u and v in T. Consider a spanning tree T in primal graph G. It is well-known that the edges not present in T form a spanning tree T^* of G^*. We call T^* the *dual tree* of T. The following lemma is a well-known result [30].

Lemma 1. *For an edge e in primal tree T, let f and g be the faces to either side of e in G. Then the edges of $E \setminus \{e\}$ that connect the two subtrees of $T \setminus \{e\}$ are exactly those on the simple path in T^* from f to g.*

A co-path is a sequence of faces that is a sequence of vertices in the dual that form a path. A co-path *avoids* a set of edges F if every consecutive pair of faces in the sequence shares an edge not in F.

3 Edge Failures

In this section, we develop a data structure that, given a planar undirected graph $G = (V, E)$, an integer $d \geq 1$, and a dynamic subset of at most d *failed edges*, supports the following operations:

1. `update`(F): set F to be the set of failed edges;
2. `connected?`(u, v): are vertices u and v connected in $G \setminus F$?

We may assume that G is plane and connected. We will examine this problem in the dual by way of the following lemmas:

Lemma 2. *Two vertices u and v are connected in $G \setminus F$ iff there is a u-to-v co-path in G^* avoiding F.*

Proof. If u and v are connected in $G \setminus F$, then there is a u-to-v path P in G not using any edges in F. This path, viewed as a sequence of vertices in G is a sequence of faces, or co-path, in G^*. Since consecutive vertices in P are adjacent by way of edges not in F, consecutive faces in the identified co-path share an edge not in F: this u-to-v co-path avoids F.

Conversely, let P be a u-to-v co-path avoiding F. P is a sequence of faces in G^*, and so is a sequence of vertices in G. Consecutive faces on P in G^* share an edge avoiding F and so are adjacent by way of edges not in F: P is a path in G not using any edges in F.

Consider the subgraph G^*_F of G^* consisting of the failed edges, F. Let G^*_F inherit the embedding of G^*. We refer to the faces of G^*_F as *superfaces*. Each superface corresponds to the union of faces of G^*.

Lemma 3. *Let f and g be faces of G^*. There is an f-to-g co-path in G^* avoiding F iff f and g are contained in the same superface of G^*_F.*

Proof. Suppose f and g are contained in different superfaces; let C be the set of edges that bound the superface containing f. Any f-to-g co-path requires a face on either side of C and so cannot avoid C, which is a subset of F.

Suppose otherwise; let S be the set of faces of G^* that form the superface f_S containing f and g. Since f_S is a face of G^*_F, there is a curve $\mathcal{C}$ contained entirely in f_S that starts inside f and ends inside g. Consider the sequence of faces in S that this path visits; this is an F-avoiding f-to-g co-path as consecutive faces must share an edge that $\mathcal{C}$ crosses and this edge cannot be in F.

In light of Lemma 3 we can restate the operations as follows:

1. `update`(F): set F to be the set of failed edges;
2. `connected?`(u, v): are u and v contained in the same superface of G^*_F?

3.1 The Data Structure

Fix a rooted spanning tree T of the primal graph G. We use T to determine the superfaces of G^*_F containing the faces corresponding to the query vertices u and v. To do so, we require constant-time lca query support for T [24,2,7].

update(F)*:* The edges F are listed in no particular order. We start by building the planar embedding of the subgraph G^*_F induced by F that is inherited from G^*. Let $V(F)$ be the endpoints of F (in the dual sense). For each vertex $v \in V(F)$ identify the edges of F incident to v and their cyclic ordering[3] in G^*. We can compute these orderings in sort(d, n) time.

The boundaries of the superfaces given by G^*_F can be traversed in $O(d)$ time given this combinatorial embedding. The set $V(F)$ is the set of faces of G^* that are along the boundaries of superfaces of G^*_F. Label a dual face/primal vertex in $V(F)$ with the superface that contains it; mark the vertices of the static tree T with these superface labels.

[3] The cyclic ordering of edges incident to each vertex is sufficient to define the embedding. [16]

connected?(u, v)*:* To answer a query $\texttt{connected?}(u, v)$, suppose we have identified the first and last marked vertices (if any) on the u-to-v path in T; call them $\hat{u}$ and $\hat{v}$.[4] By Lemmas 2 and 3, u and v are connected in $G \setminus F$ iff $\hat{u}$ and $\hat{v}$ do not exist, or $\hat{u}$ and $\hat{v}$ are labelled with the same superface. Therefore, we need only identify $\hat{u}$ and $\hat{v}$, if they exist. Lemma 4 shows that finding $\hat{u}$ and $\hat{v}$ is reducible to one least common ancestor query and $O(1)$ predecessor queries.

Lemma 4. *Let T be a tree of size n and $M \subseteq V(T)$ be a set of* marked *vertices, with $|M| = d$. Then after $O(n)$ preprocessing (independent of M), an $O(d)$-size data structure can be constructed in $O(\mathrm{sort}(d, n))$ time that answers the following query in $O(\mathrm{pred}(d, n))$ time: Given $u, v \in V(T)$, what are the first and last M-vertices on the u-v path?*

Proof. Recall that T is rooted. Let $w = \mathrm{lca}(u, v)$. The first marked vertex on the u-to-v path is either the first marked vertex on the u-to-w path or the last marked vertex on the v-to-w path. Thus, we can assume without loss of generality that v is an ancestor of u. Fix an arbitrary DFS of T and let $\mathrm{pre}(x)$ (resp. $\mathrm{post}(x)$) be the time when x is pushed onto (resp. popped off) the stack during DFS. Given M, we first sort the pre and post indices of its elements, in $O(\mathrm{sort}(d, n))$ time, which allows us to label each $x \in M$ with the nearest strictly ancestral M-vertex $\mu(x)$. Here it is convenient to assume that the root is marked (honorarily, if not in M) so $\mu(x)$ is defined everywhere except the root. We build a global predecessor structure on the set $\mathcal{S} = \{\mathrm{pre}(x), \mathrm{post}(x) \; : \; x \in M\}$ and local predecessor structures on the sets $\mathcal{S}_x = \{\mathrm{pre}(x') \; : \; x' \in \mu^{-1}(x)\}$, where $x \in M$ and $\mathcal{S}_x$ is the set of "immediate" descendants in M connected by a path of non-M vertices. To answer a query u, v (where v is ancestral to u) we first find the closest marked ancestors $u', v' \in M$ as follows. Let i be the predecessor of $\mathrm{pre}(u)$ in $\mathcal{S}$. If $i = \mathrm{pre}(y)$ for some $y \in M$ then x is an immediate descendant of y and $u' = y$. If $i = \mathrm{post}(y)$ then let $u' = \mu(y)$. See Figure 1(a). It follows that u' is an ancestor of u and that there are no other M-vertices on the path from u to u'. If u' is ancestral to v then there are no marked nodes on the u-v path, so assume this is not the case. In order to find the *last* marked vertex on the path from u to v we search for the predecessor of $\mathrm{pre}(u)$ in $\mathcal{S}_{v'}$, say $\mathrm{pre}(y)$. Since there is *some* marked vertex on the path from u to v' it follows that u is a descendant of y, which is a descendant of v and that there are no other marked vertices on the path from y to v. See Figure 1(b).

We now have the following:

Theorem 1. *There is a data structure for planar undirected n-vertex graphs and any $d \geq 1$ that after $O(n)$ preprocessing time supports* $\textbf{\textit{update}}(F)$ *in $O(\mathrm{sort}(d, n))$ time, where $F \subseteq E$, $|F| \leq d$, and supports* $\textbf{\textit{connected?}}(u, v)$ *in $O(\mathrm{pred}(d, n))$ time.*

[4] This is a slightly more general problem than the *marked ancestor* problem considered by Alstrup, Husfeldt, and Rauhe [3].

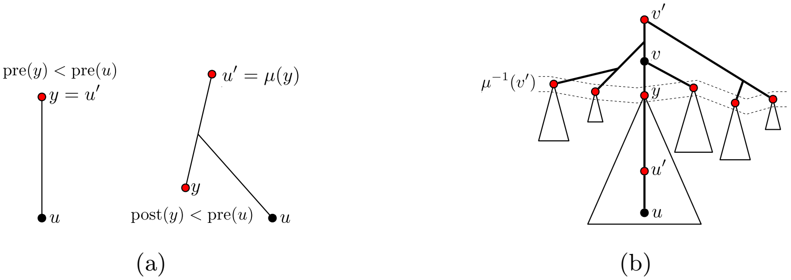


Fig. 1. Marked vertices are shaded red. (a) The nearest marked ancestor u' of u depends on whether the predecessor of pre(u) is pre(y) (left) or post(y) (right) for some marked y. (b) v' is the nearest marked ancestor of v. The last marked ancestor y on the u-to-v path is in $\mu^{-1}(v')$.

4 Vertex and Edge Failures

We now turn our attention to the scenario where both edges and vertices can fail. The additional challenge arises from high degree vertices that, when failed, can greatly reduce the connectivity of the graph. Nevertheless we will show how to maintain dynamic connectivity in time proportional to the number of failed vertices, rather than their degrees. Formally, we develop a data structure that, given a planar, undirected graph $G = (V, E)$, an integer $d \geq 1$, and a dynamic subset of at most d failed edges and vertices, supports the following operations:

1. `update`(F): set F to be the failed set of vertices and edges;
2. `connected?`(u, v): are vertices u and v connected in $G \setminus F$?

4.1 Vertex Failures for Triconnected Planar Graphs

In this subsection we shall assume that G is triconnected. This allows us to apply Barnette's theorem, which states that every triconnected planar graph has a spanning tree of degree at most three [6]. Furthermore, such a degree-three tree, T, can be found in linear time [28]. In Section 6 we show that this assumption is basically without loss of generality: we can reduce the problem on general graphs to triconnected graphs with an additive pred(d, n) slowdown in the query and update algorithms.

Assume that only vertices fail. The full manuscript [9] describes how to handle both vertex and edge failures. Let C be the clockwise cycle that bounds the infinite and only face of T. C is an embedding-respecting Euler tour of T that visits each edge twice and each vertex at most three times. For a non-empty subset F of V, partition C into maximal subpaths whose internal nodes are not in F. Denote this set of paths by $\mathcal{P}_F$. Note that $|\mathcal{P}_F| \leq 3|F|$ and therefore that a connected component of $T \backslash F$ is made up of possibly many paths in $\mathcal{P}_F$. Assign the connected components of $T \setminus F$ distinct colors and label each path in $\mathcal{P}_F$ with the color of its component.

Let e_1 and e_2 be the edges before and after a particular copy of vertex v in the order given by C. Let f_i be the face of G to the left of e_i. Root the dual spanning tree T^* at the infinite face of G and let $\ell = \mathrm{lca}_{T^*}(f_1, f_2)$. If ℓ is not the root of T^*, let e_v be the parent edge of ℓ in T^*. Let L be the set of edges e_v (if well defined) with neither endpoint in F for all failed vertices $v \in F$ (according to their multiplicity in C). Note that $|L| \le 3d$. By duality, we consider L as a subset of primal edges. Considered as an edge of the primal graph, e_v forms a cycle with T that v is on; that is, e_v witnesses an alternate connection should v fail.

We define an auxiliary graph H_F that will succinctly represent the connectivity of $\mathcal{P}_F$. The nodes of H_F are the paths in $\mathcal{P}_F$. Two path-nodes $P_1, P_2 \in \mathcal{P}_F$ are adjacent in H_F iff

– P_1 and P_2 have the same color and are consecutive paths in C among paths of the same color, or
– there is an edge in L between the interior of P_1 and the interior of P_2.

There are at most $|\mathcal{P}_F|$ edges of the first type and $|L|$ edges of the second type. Since $|\mathcal{P}_F| \le 3d$ and $|L| \le 3d$, $|H_F| = O(d)$.

Lemma 5. *Paths in $\mathcal{P}_F$ are connected in $G \setminus F$ iff they are connected in H_F.*

Proof. Consider distinct paths P_1 and P_2 in $\mathcal{P}_F$. It is clear from the definition of L and H_F that if there is an edge (P_1, P_2) in H_F then the interior of P_1 and the interior of P_2 are connected in $G \setminus F$. This implies the "if" part.

Since path-nodes of a given color class are connected by a cycle in H_F given by the paths order along C. If two paths are of the same color, their path-nodes will be connected in H_F. So, for the "only if" part, it suffices to show that if P_1 and P_2 are different colors but (by transitivity of connectivity) there is a single edge $e \in G \setminus F$ between the interior of P_1 and the interior of P_2, then they are connected in H_F.

So let $e = (u_1, u_2)$ be such an edge where $u_i \in P_i$. Let f_1 and f_2 be the faces incident to e in G such that f_1 is the child of f_2 in T^*. Let f be the bounded face of $T \cup \{e\}$ (which contains e and encloses f_1). Let v_i be the failed vertex that is an endpoint of P_i on the bounded face of $T \cup \{e\}$.

We continue by induction on the number of faces $k \ge 1$ of G contained in f. In the base case $k = 1$ and $f = f_1$. Here $e_{v_1} = e_{v_2} = e$, so $e \in L$ and (P_1, P_2) is an edge of H_F. Now assume $k > 1$ and that the inductive hypothesis holds for smaller values.

If f_1 has a failed vertex v on its boundary, then we argue $e_v = e$ (and so (P_1, P_2) is an edge of H_F). Consider the copy of v in the traversal of C that has f_1 to the left. Let g_1 and g_2 be the faces to the left of the edges before and after this copy of v in C. Since $e \in T^*$, e is an ancestral edge of g_1 and g_2. Since g_1, g_2, f_1 all contain v as a boundary vertex, $\mathrm{lca}_{T^*}(g_1, g_2) = f_1$.

Suppose that f_1 has no failed vertices on the boundary. For each edge $e' \ne e$ on the boundary of f_1, if $e' \in T$ then e' belongs to a path of $\mathcal{P}_F$. Otherwise, the bounded face of $T \cup \{e'\}$ is contained in f and does not contain f_1 so it contains fewer than k faces of G. Then the paths of $\mathcal{P}_F$ containing the endpoints of e' in their interior are connected in H_F if they have distinct colors (by the inductive argument) or if they have the same color (by the start of this proof). Since this holds for every edge e' of $f_1 \setminus \{e\}$, P_1 and P_2 are connected in H_F, as desired.

4.2 The Data Structure

In a linear-time preprocessing step, we compute T, T^*, and C and initialize an lca data structure.

`update`(F)*:* We build H_F for the input set of failed vertices F. Let f be the sum of the degrees in T of the vertices in F; note, $f \leq 3d$. Let F' be the multiset of failed vertices according to their multiplicity along C, i.e., their degree in T. Again, $|F'| = f$. Sort the vertices of F' according to their order along C. This provides an implicit representation of $\mathcal{P}_F$. Label these paths according to their ordinal in F'; that is, path P_i is the path starting with the i^{th} vertex in sorted F'. This takes time $\text{sort}(f, n)$.

Greedily assign colors to the paths, considering the paths in order. In each iteration we assign colors to all the paths in a given color class. Upon considering path P_i, if P_i has not yet been colored, assign it a new color. Check the second last endpoint of P_i and find the edge e that precedes that vertex in C *after* P_i. Let P_j be the path that starts with edge e (which can be found by e's starting point in C). Color P_j with same color as P_i. Repeat until returning to P_i. This takes time $O(f)$.

Build the set L of edges. For each edge in L, identify the paths in $\mathcal{P}_F$ that contain its endpoints. We do this in bulk. Sort the set of endpoints of L along with F', that is, $V(L) \cup F'$, according to their order along C. Traverse this order, assigning each endpoint in $V(L)$ the path corresponding to the last failed vertex visited. This takes time $\text{sort}(f, n)$.

From L (with endpoints labelled with the appropriate path in $\mathcal{P}_F$) and the colors of $\mathcal{P}_F$, build H_F. Compute the connected components of H_F and label the path-nodes of H_F with the name of the connected component it belongs to. This takes time $O(f)$.

The total time for `update` is bounded by $\text{sort}(f, n) = O(\text{sort}(d, n))$.

`connected?`(u, v)*:* We may assume $u, v \notin F$. In $O(\text{pred}(d, n))$ time identify any path P_u containing u in the interior and any path P_v containing v in the interior. By Lemma 5, u and v are connected in $G \setminus F$ iff P_u and P_v are in the same connected component of H_F. The latter condition is checked in constant time given the labels of the connected components.

5 Decremental Subgraph Connectivity

In this section, we consider the model in which updates and queries are intermixed. We only allow failures and not recoveries but no longer assume an upper bound d on $|F|$, that is, F is a growing set.

As in Section 4, we first assume that G is triconnected and that only vertices can fail. Define T, T^*, C, L, and F as before and build an lca data structure for T^*. Initially L and F are empty. Redefine H_F to be $(V, T \setminus F \cup L)$, that is, we no longer contract paths in $\mathcal{P}_F$. For each vertex v we maintain a list $L(v)$ of the lca-edges L incident to v. Represent L as a subset of marked edges of G.

When a vertex v fails, unmark $L(v)$ in L since L should only contain edges not incident to failed vertices. For each of the at-most-three occurrences of v along C, find the corresponding lca-edge e_v (if it exists); if e_v has no endpoint in F, mark e_v as a member of L and add e_v to the lists of its endpoints.

It follows easily from the definition of L and of H_F that both L and H_F are updated correctly after each failure. Lemma 5 (that two vertices are connected in $G \setminus F$ iff they are connected in H_F) holds with these definitions; the only difference is the interior of the paths from $\mathcal{P}_F$ are explicitly represented with the edges from $T \setminus F$. Note that showing the requisite connectivity for Lemma 5 does not depend on lca-edges that have a failed endpoint. Removing (unmarking) the edges $L(v)$ as members of L when v fails is equivalent to not including them in L as used in the proof of Lemma 5.

We maintain H_F with an oracle which allows us to delete an edge and an isolated vertex in $O(\log n)$ time [18] since a vertex failure results in at most three edges of T being deleted from H_F. Furthermore, at most three edges are added to L and each of these edges is added only to the two lists associated with its endpoints. Hence if F is the final set of deleted vertices, then a total of $O(|F|)$ edges are added or removed from H_F and there are at most $O(|F|)$ updates to L and to the lists $L(v)$. It follows that each deletion takes $O(\log n)$ amortized time. A connectivity query can be answered in $O(\log n)$ worst-case time with the oracle associated with H_F.

So far we considered only vertex deletions. The full manuscript [9] details how to handle both vertex and edge deletions. Using the reduction from general planar graphs to triconnected planar graphs in Theorem 4 we have the following.

Theorem 2. *There is a data structure for decremental subgraph connectivity in triconnected planar graphs that, after $O(n)$ preprocessing time, allows an intermixed sequence of vertex/edge failures and connectivity queries in $O(\log n)$ amortized time per failure (i.e., f failures in $O(f \log n)$ time total) and $O(\log n)$ worst-case time per query. In general planar graphs there is a data structure requiring $O(\log^2 n)$ amortized time per failure and $O(\log^2 n)$ worst-case time per query.*

6 Reductions to Triconnected Graphs

In this section we state the reductions from general (planar) graphs to triconnected (planar) graphs that were used in Sections 4 and 5. Although the algorithm from Section 5 handles only failures, the reduction allows both failures and recoveries. See the full manuscript for proofs [9].

Theorem 3. (The d-Failure Model) *Suppose there is a connectivity oracle $\mathcal{A}_3$ for any triconnected (planar) graph G with the following parameters. The preprocessing time is $P_3(n,m)$, the time to update the structure after d edge or vertex failures is $U_3(d,n)$, and the time for a connectivity query is $Q_3(d,n)$. Then there is a connectivity oracle $\mathcal{A}$ for any (planar) graph G with parameters $P = O(P_3)$, $U = O(U_3 + \mathrm{sort}(d,n) + d \cdot (Q_3 + \mathrm{pred}(d,n)))$, and $Q = O(Q_3 + \mathrm{pred}(d,n))$.*

Theorem 4. (The Fully Dynamic Subgraph Model) *Suppose there is a connectivity oracle $\mathcal{A}_3$ for any triconnected (planar) graph G with the following parameters. The preprocessing time is $P_3(n,m)$, the time to process a vertex failure or recovery is $U_3^v(n)$, the time to process an edge failure or recovery $U_3^e(n)$, and the time for a connectivity query is $Q_3(n)$. Then there is a connectivity oracle $\mathcal{A}$ for any (planar) graph G with parameters $P = O(P_3)$, $U^e = O((U_3^e + Q_3 + \log n/\log\log)\log n)$, $U^v = O(U_3^v + (U_3^e + Q_3 + \log n/\log\log n)\log n)$, and $Q = O((Q_3 + \log n/\log\log n)\log n)$.*

References

1. Afshani, P., Chan, T.M.: Dynamic connectivity for axis-parallel rectangles. Algorithmica 53(4), 474–487 (2009)
2. Alstrup, S., Gavoille, C., Kaplan, H., Rauhe, T.: Nearest common ancestors: a survey and a new distributed algorithm. In: Proceedings 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 258–264 (2002)
3. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: Proceedings 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 534–544 (1998)
4. Andersson, A., Hagerup, T., Nilsson, S., Raman, R.: Sorting in linear time? J. Comput. Syst. Sci. 57(1), 74–93 (1998)
5. Andersson, A., Thorup, M.: Dynamic ordered sets with exponential search trees. J. ACM 54(3), 13 (2007)
6. Barnette, D.: Trees in polyhedral graphs. Canadian Journal of Mathematics 18, 731–736 (1966)
7. Bender, M.A., Farach-Colton, M.: The LCA Problem Revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
8. Bernstein, A., Karger, D.: A nearly optimal oracle for avoiding failed vertices and edges. In: Proceedings 41st Annual ACM Symposium on Theory of Computing (STOC), pp. 101–110 (2009)
9. Borradaile, G., Pettie, S., Wulff-Nilsen, C.: Connectivity Oracles for Planar Graphs. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 316–327. Springer, Heidelberg (2012)
10. Chan, T.: Dynamic subgraph connectivity with geometric applications. SIAM J. Comput. 36(3), 681–694 (2006)
11. Chan, T.M., Patrascu, M., Roditty, L.: Dynamic connectivity: Connecting to networks and geometry. SIAM J. Comput. 40(2), 333–349 (2011)
12. Demetrescu, C., Thorup, M., Chowdhury, R.A., Ramachandran, V.: Oracles for distances avoiding a failed node or link. SIAM J. Comput. 37(5), 1299–1318 (2008)

13. Duan, R.: New Data Structures for Subgraph Connectivity. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 201–212. Springer, Heidelberg (2010)
14. Duan, R., Pettie, S.: Dual-failure distance and connectivity oracles. In: Proceedings 20th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 506–515 (2009)
15. Duan, R., Pettie, S.: Connectivity oracles for failure prone graphs. In: Proceedings 42nd ACM Symposium on Theory of Computing, pp. 465–474 (2010)
16. Edmonds, J.: A combinatorial representation for polyhedral surfaces. Notices of the American Mathematical Society 7, 646 (1960)
17. Eppstein, D., Galil, Z., Italiano, G., Nissenzweig, A.: Sparsification — a technique for speeding up dynamic graph algorithms. J. ACM 44(5), 669–696 (1997)
18. Eppstein, D., Italiano, G.F., Tamassia, R., Tarjan, R.E., Westbrook, J., Yung, M.: Maintenance of a minimum spanning forest in a dynamic plane graph. J. Algor. 13(1), 33–54 (1992)
19. Frederickson, G.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. 14(4), 781–798 (1985)
20. Fredman, M.L., Willard, D.E.: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. J. Comput. Syst. Sci. 48(3), 533–551 (1994)
21. Frigioni, D., Italiano, G.F.: Dynamically switching vertices in planar graphs. Algorithmica 28(1), 76–103 (2000)
22. Han, Y.: Deterministic sorting in $O(n \log\log n)$ time and linear space. J. Algorithms 50, 96–105 (2004)
23. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proceedings of the 43rd Symposium on Foundations of Computer Science, FOCS 2002, pp. 135–144. IEEE Computer Society, Washington, DC (2002)
24. Harel, D., Tarjan, R.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)
25. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM 48(4), 723–760 (2001)
26. Pătraşcu, M., Thorup, M.: Time-space trade-offs for predecessor search. In: Proceedings 38th ACM Symposium on Theory of Computing (STOC), pp. 232–240 (2006)
27. Pătraşcu, M., Thorup, M.: Planning for fast connectivity updates. In: Proceedings 48th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 263–271 (2007)
28. Strothmann, W.: Bounded degree spanning trees. PhD thesis, Universität-Gesamthochschule Paderborn (1997)
29. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Math. Syst. Theory 10, 99–127 (1977)
30. Whitney, H.: Planar graphs. Fundamenta mathematicae 21, 73–84 (1933)
31. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space $\theta(n)$. Info. Proc. Lett. 17(2), 81–84 (1983)

Access Graphs Results for LRU versus FIFO under Relative Worst Order Analysis*

Joan Boyar, Sushmita Gupta, and Kim S. Larsen

University of Southern Denmark, Odense, Denmark
{joan,sgupta,kslarsen}@imada.sdu.dk

Abstract. Access graphs, which have been used previously in connection with competitive analysis to model locality of reference in paging, are considered in connection with relative worst order analysis. In this model, FWF is shown to be strictly worse than both LRU and FIFO on any access graph. LRU is shown to be strictly better than FIFO on paths and cycles, but they are incomparable on some families of graphs which grow with the length of the sequences.

1 Introduction

The term *online* algorithm [4] is used for an algorithm that receives its input as a sequence of items, one at a time, and for every item, before knowing the subsequent items, must make an irrevocable decision regarding the current item.

The most standard measure of quality of an online algorithm is *competitive analysis* [17,22,20]. This is basically the worst case ratio between the performance of the online algorithm compared to an optimal offline algorithm which is allowed to know the entire input sequence before processing it and is assumed to have unlimited computational power.

Though this measure is very useful and has driven much research, researchers also observed problems [22] with this measure from the beginning: many algorithms obtain the same (poor) ratio, while showing quite different behavior in practice. The *paging problem* is one of the prime examples of these difficulties. The paging problem is the problem of maintaining a subset of a large number of pages in a much smaller, faster cache with space for a limited set of k pages. Whenever a page is requested, it must be brought into cache if it is not already there. In order to make room for such a page, another page currently in cache must be evicted. Therefore, an online algorithm for this problem is often referred to as an eviction strategy.

For a number of years, researchers have worked on refinements or additions to competitive analysis with the aim of obtaining separations between different algorithms for solving an online problem. Some of the most obvious and well-known paging algorithms are the eviction strategies LRU (Least-Recently-Used) and FIFO (First-In/First-Out). One particularly notable result has been the

* Partially supported by the Danish Council for Independent Research.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 328–339, 2012.

separation of LRU and FIFO via *access graphs*. Access graphs were introduced in [5] with the aim of modelling the locality of reference that is often seen in real-life paging situations [10,11]. An access graph is an undirected graph with all pages in slow memory as vertices. Given such a graph, one then restricts the analysis of the performance of an algorithm to sequences respecting the graph, in the sense that any two distinct, consecutive requests must be neighbors in the graph. Important results in understanding why LRU is often observed to perform better than FIFO in practice were obtained in [5,9], showing that on some access graphs, LRU is strictly better than FIFO, and on no access graph is it worse; all these previous results are with respect to competitive analysis.

Attempts have been made to define new generally-applicable performance measures and to apply measures defined to solve one particular problem more generally to other online problems. A collection of alternative performance measures is surveyed in [12]. Of the alternatives to competitive analysis, *relative worst order analysis* [6] and *extra resource analysis* [19] are the ones that have been successfully applied to most different online problems. See [13] for examples of online problems and references to relative worst order analysis results resolving various issues that are problematic with regards to competitive analysis.

Paging has been investigated under relative worst order analysis in [7]. Separations were found, but LRU and FIFO were proven equivalent, possibly because locality of reference is necessary to separate these two paging algorithms. We apply the access graph technique to relative worst order analysis. Note that the unrestricted analysis in [7] corresponds to considering a complete access graph.

Overall, our contributions are the following. Using relative worst order analysis, we confirm the competitive analysis result [5] that LRU is better than FIFO for path access graphs. Since these two quality measures are so different, this is a a strong indicator of the robustness of the result. Then we analyze cycle access graphs, and show that with regards to relative worst order analysis, LRU is strictly better than FIFO. Note that this does not hold under competitive analysis. The main technical contribution is the proof showing that on cycles, with regards to relative worst order analysis, FIFO is never better than LRU. Clearly, paths and cycles are the two most fundamental building blocks, and future detailed analyses of any other graphs type will likely build on these results.

The standard example of a very bad algorithm with the same competitive ratio as LRU and FIFO is FWF, which is shown to be strictly worse than both LRU and FIFO on any access graph (containing a path of length at least $k+1$), according to relative worst order analysis. Using relative worst order analysis, one can often obtain more nuanced results. This is also the case here for general access graphs, where we establish an incomparability result.

None of the algorithms we consider require prior knowledge of the underlying access graph. This issue was pointed out in [15] and [16] in connection with the limitations of some of the access graph results given in [5,14,18] and the Markov paging analogs in [21].

As relative worst order analysis is getting more established as a method for analyzing online algorithms, it is getting increasingly important that the theoretical toolbox is extended to match the options available when carrying out competitive analysis. Recently, in [13], list factoring [1,3] was added as an analytical tool when using relative worst order analysis on list accessing problems [22,2], and here we demonstrate that access graphs can also be included.

After a preliminary section, we prove that LRU is never worse than FIFO on paths or cycles. Then we establish separation results, showing that LRU is strictly better than FIFO on paths and cycles of length at least $k + 1$ and that both algorithms are strictly better than FWF on any graph containing a path of length at least $k+1$. The last result proves the incomparability of LRU and FIFO on general access graphs, using a family of graphs where the size is proportional to the length of the request sequence. We conclude with some open problems regarding determining completely for which classes of graphs LRU is better than FIFO. Some of the proofs have been omitted due to space constraints. They can be found in the full version [8].

2 Preliminaries

The *paging problem* is the problem of processing a sequence of page requests with the aim of minimizing the number of page faults. Pages reside in a large memory of size N, but whenever a page is requested, it must also be in the smaller cache of size $k < N$. If it is already present, we refer to this as a *hit*. Otherwise, we have a *fault* and must bring the page into cache. Except for start-up situations with a cache that is not full, this implies that some page currently in cache must be chosen to be evicted by a paging algorithm.

If $\mathbb{A}$ is a paging algorithm and I an input sequence, we let $\mathbb{A}(I)$ denote the number of faults that $\mathbb{A}$ incurs on I. This is also called the *cost* of $\mathbb{A}$ on I.

An important property of some paging algorithms that is used several times in this paper is the following:

Definition 1. *An online paging algorithm is called* conservative *if it incurs at most k page faults on any consecutive subsequence of the input containing k or fewer distinct page references.*

The algorithms, Least-Recently-Used (LRU) and First-In/First-Out (FIFO) are examples of conservative algorithms. On a page fault, LRU evicts the least recently used page in cache and FIFO evicts the page which has been in cache the longest. Flush-When-Full (FWF), which is not conservative, evicts all pages in cache whenever there is a page fault and its cache is full.

An input sequence of page requests is denoted $I = \langle r_1, r_2, \ldots, r_{|I|} \rangle$. We use standard mathematical interval notation to denote subsequences. They can be open, closed, or semi-open, and are denoted by (r_a, r_b), $[r_a, r_b]$, $(r_a, r_b]$, or $[r_a, r_b)$. If S is a set of pages, we call a request interval S-*free* if the interval does not contain requests to any elements of S. We use the following notation for graphs.

Definition 2. *The path graph on N vertices is denoted P_N and a cycle graph on N vertices is denoted C_N. A walk is an ordered sequence of vertices where* *consecutive vertices are either identical or adjacent in the graph. A path is a walk in which every vertex appears at most once. The length of a walk $\mathcal{W}$ is the number of (not necessarily distinct) vertices in it, denoted by $|\mathcal{W}|$. The set of distinct vertices in a walk $\mathcal{W}$ is denoted by $\{\mathcal{W}\}$.*

Definition 3. *An* access graph $G = (V, E)$ *is a graph whose vertex set corresponds to the set of pages that can be requested in a sequence. A sequence is said to* respect *an access graph, if the sequence of requests constitutes a walk in that access graph.*

In the relative worst order analyses carried out in this paper, permutations play a key role. We introduce some notation for this and then present the standard definition of the relative worst order quality measure.

For an algorithm $\mathbb{A}$, $\mathbb{A}_W(I)$ is the cost of the algorithm $\mathbb{A}$ on the worst reordering of the input sequence I, i.e., $\mathbb{A}_W(I) = \max_\sigma \mathbb{A}(\sigma(I))$, where σ is a permutation on $|I|$ elements and $\sigma(I)$ is a reordering of the sequence I.

Definition 4. *For any pair of paging algorithms $\mathbb{A}$ and $\mathbb{B}$, we define*

$$\begin{aligned} c_l(\mathbb{A},\mathbb{B}) &= \sup\{c \mid \exists b\colon\ \forall I\colon\ \mathbb{A}_W(I) \geq c\,\mathbb{B}_W(I) - b\} \text{ and} \\ c_u(\mathbb{A},\mathbb{B}) &= \inf\{c \mid \exists b\colon\ \forall I\colon\ \mathbb{A}_W(I) \leq c\,\mathbb{B}_W(I) + b\} \end{aligned}$$

If $c_l(\mathbb{A},\mathbb{B}) \geq 1$ or $c_u(\mathbb{A},\mathbb{B}) \leq 1$, the algorithms are said to be comparable *and the* relative worst order ratio $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *of algorithm $\mathbb{A}$ to $\mathbb{B}$ is defined. Otherwise,* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *is undefined. If $c_l(\mathbb{A},\mathbb{B}) \geq 1$, then $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = c_u(\mathbb{A},\mathbb{B})$ and if $c_u(\mathbb{A},\mathbb{B}) \leq 1$, then $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = c_l(\mathbb{A},\mathbb{B})$.*

If $\mathrm{WR}_{\mathbb{A},\mathbb{B}} < 1$, algorithms $\mathbb{A}$ and $\mathbb{B}$ are said to be comparable in $\mathbb{A}$'s favor. *Similarly, if $\mathrm{WR}_{\mathbb{A},\mathbb{B}} > 1$, the algorithms are said to be comparable in $\mathbb{B}$'s* favor.

When we use this measure to compare algorithms on a given access graph G, we use the notation $\mathbb{A}_W^G(I)$ to denote the cost of $\mathbb{A}$ on a worst permutation of I that respects G. Similarly, we use $\mathrm{WR}_{\mathbb{A},\mathbb{B}}^G$ to denote the relative worst order ratio of algorithms $\mathbb{A}$ and $\mathbb{B}$ on the access graph G.

Finally, let $\mathrm{Worst}(I, G, \mathbb{A})$ denote the set of worst orderings for the algorithm $\mathbb{A}$ of I respecting the access graph G, i.e., any sequence in $\mathrm{Worst}(I, G, \mathbb{A})$ is a permutation of I respecting G, and for any $I \in \mathrm{Worst}(I, G, \mathbb{A})$, $\mathbb{A}(I) = \mathbb{A}_W^G(I)$.

3 Paths

In [5, Theorem 13], it has been shown that if the access graph is a tree, then LRU is optimal among all online algorithms. Furthermore, in the case of path graphs, LRU matches the performance of an optimal offline algorithm.

Theorem 1. *For all I respecting the path P_N, $\mathrm{LRU}_W^{P_N}(I) \leq \mathrm{FIFO}_W^{P_N}(I)$.*

Proof. Consider any sequence I respecting P_N. Let I' be a worst ordering for LRU among the permutations of I respecting P_N. Then, using LRU's optimality on trees for the first inequality, $\mathrm{LRU}_W^{P_N}(I) = \mathrm{LRU}(I') \leq \mathrm{FIFO}(I') \leq \mathrm{FIFO}_W^{P_N}(I)$. □

4 Cycles

Almost this entire section is leading up to a proof that for all I respecting the access graph C_N, $\mathrm{LRU}_W^{C_N}(I) \leq \mathrm{FIFO}_W^{C_N}(I)$. Notice that this theorem is not trivial, since there exist sequences respecting the cycle access graph where FIFO does better than LRU. Consider, for example, the cycle on four vertices $C_4 = \langle 1, 2, 3, 4 \rangle$, $k = 3$, and the request sequence $I = \langle 2, 1, 2, 3, 4, 1 \rangle$. With this sequence, at the request to 4, LRU evicts 1 and FIFO evicts 2. Thus, FIFO does not fault on the last request and has one fault fewer than LRU. Note that on the reordering, $I' = \langle 1, 2, 2, 3, 4, 1 \rangle$, LRU still faults five times, but FIFO does too. This is the transformation which would be performed in Lemma 3 below, combined with the operation in the proof of Lemma 1 to reinsert requests which have been removed. Note that this is not a worst ordering for LRU, since LRU and FIFO both fault six times on $I'' = \langle 1, 2, 3, 4, 1, 2 \rangle$.

Each of the results leading up to the main theorem in this section is aimed at establishing a new property that we may assume in the rest of the section. Formally, these results state that if we can prove our end goal *with* the new assumption, then we can also prove it without. Thus, it is just a formally correct way of phrasing that we are reducing the problem to a simpler one. Some of the sequence transformations we perform in establishing these properties also remove requests, in addition to possibly reordering. The following general lemma allows us to do this in all of these specific cases.

Lemma 1. *Assume we are given an access graph G, a sequence I respecting G, and a sequence $I_{\mathrm{LRU}} \in \mathrm{Worst}(I, G, \mathrm{LRU})$. We write I_{LRU} as the concatenation of three subsequences $\langle I_1, I_2, I_3 \rangle$. Let I' be $\langle I_1, I'_2, I_3 \rangle$, where I'_2 can be any subsequence (not necessarily of the same length as I_2) such that I' still respects G. Assume that* LRU *incurs at least as many faults on I'_2 as on I_2, and the cache content, including information concerning which pages are least recently used, is exactly the same just after I'_2 in I' as after I_2 in I_{LRU}. Assume further that I'_2 is obtained from I_2 by removing some requests and/or reordering requests, and that $\{I\} = \{I'\}$. Then, $I' \in \mathrm{Worst}(I', G, \mathrm{LRU})$, and if $\mathrm{LRU}(I') \leq \mathrm{FIFO}_W^G(I')$, then $\mathrm{LRU}_W^G(I) \leq \mathrm{FIFO}_W^G(I)$.*

By repeatedly removing the $j - 1$ hits in a sequence of j consecutive requests to the same page, we establish the following property:

Property 1. In proving for any access graph G, any sequence I respecting G, and any $I_{\mathrm{LRU}} \in \mathrm{Worst}(I, G, \mathrm{LRU})$ that $\mathrm{LRU}(I_{\mathrm{LRU}}) \leq \mathrm{FIFO}_W^G(I)$, we may assume that I_{LRU} has no consecutive requests to the same page.

We give a collection of definitions enabling us to describe how a request sequence without consecutive requests to the same page behaves on the cycle.

Definition 5.

- *An arc is a connected component of a cycle graph. As a mathematical object, an arc is the same as a path (in this section), but refers to a portion of C_N, rather than a part of the walk defined by a request sequence.*

- *One can fix an orientation in a cycle so that the concepts of moving in a clockwise or anti-clockwise direction are well-defined. We refer to a walk as being* uni-directional *if each edge is traversed in the same direction as the previous, and abbreviate this* u-walk.
- *A request r_i in the request sequence is a* turn *if the direction changes at that vertex, i.e., if r_i is neither the first nor the last request and $r_{i-1} = r_{i+1}$. The vertex requested is referred to as a* turning point.
- *When convenient we will represent a request sequence I by its* turn sequence,
$$T = \langle A_1, v_1, A_2, v_2, \dots, A_z, v_z \rangle,$$
where $T = I$, v_z is simply the last request of the sequence, all the other v_i's are the turns of the request sequence, and all the A_i's are u-walks. Thus, for all $i < z$, either $A_i \subseteq A_{i+1}$ or $A_{i+1} \subseteq A_i$. We refer to a turn v_i as a clockwise (anti-clockwise) turn *if the A_{i+1} goes in the clockwise (anti-clockwise) direction.*
- *Two turns are said to be* opposite *if they are in different directions.*
- *If for some $i < z$, $|A_{i+1} \cup \{v_{i+1}\}| \geq k$, then v_i is an* extreme turn. *Otherwise, v_i is a* trivial turn.

Most of the above is obvious terminology about directions around the circle. The last definition, on the other hand, is motivated by the behavior of the paging algorithms that we analyze. Not surprisingly, it turns out to be an important distinction whether or not the cache will start evicting pages before turning back. We treat this formally below.

We now reduce our problem to sequences without trivial turns.

Lemma 2. *Assume Property 1. For the access graph C_N, assume that for any I and $I_{\mathrm{LRU}} \in \mathrm{Worst}(I, C_N, \mathrm{LRU})$, where I_{LRU} has no trivial turns, we have that $\mathrm{LRU}(I_{\mathrm{LRU}}) \leq \mathrm{FIFO}_W^{C_N}(I)$. Then, for any I, $\mathrm{LRU}_W^{C_N}(I) \leq \mathrm{FIFO}_W^{C_N}(I)$.*

We have now established the following property:

Property 2. We may assume that a worst ordering for LRU has no trivial turns.

Lemma 3. *Assume Properties 1–2. For the access graph C_N, assume that for any sequence I and $I_{\mathrm{LRU}} \in \mathrm{Worst}(I, C_N, \mathrm{LRU})$, where I_{LRU} has turn sequence $\langle A_1, v_1, A_2, v_2, \dots, A_z, v_z \rangle$ and $\forall i : \ |A_i| \geq k-1$, we have that $\mathrm{LRU}(I_{\mathrm{LRU}}) \leq \mathrm{FIFO}_W^{C_N}(I)$. Then, for any I, $\mathrm{LRU}_W^{C_N}(I) \leq \mathrm{FIFO}_W^{C_N}(I)$.*

Proof. By Property 2, we may assume that there are no trivial turns. Thus, we already know that for any i, $1 < i \leq z$, the result holds.

If $|A_1| < k-1$, then we replace I_{LRU} by $I'_{\mathrm{LRU}} = \langle v_1, A_2, \dots \rangle$. This preserves the number of faults in the subsequence $\langle v_1, A_2 \rangle$ compared with $\langle A_1, v_1, A_2 \rangle$, and since $|A_2| \geq k-1$, LRU is in the same state after processing A_2 in I'_{LRU} as it was in processing I_{LRU}. By Lemma 1, we can use I'_{LRU}. □

We have now established the following property:

Property 3. We may assume that a worst ordering for LRU is of the form

$$\langle A_1, v_1, A_2, v_2, \ldots, A_z, v_z \rangle \text{ where } \forall i\colon\ |A_i| \geq k-1.$$

If the first three properties hold for some sequence, I, then it is easy to see that the number of turns determines how many hits LRU has on I.

Proposition 1. *If I has the form of Property 3 and contains no repeated requests to the same page, then* LRU *has exactly $(z-1)(k-1)$ hits on I.*

Next we show that we may assume that in a worst ordering for LRU, there is no turn which is followed by a full cycle in the opposite direction.

Definition 6. *Let u, v, and w be three distinct consecutive vertices on C_N. We refer to I as having an* overlap *if I can be written as $\langle \ldots, u, v, u, B, w, v, \ldots \rangle$. If I does not have an overlap, we refer to I as* overlap-free.

Lemma 4. *Assume Properties 1–3. For the access graph C_N, assume that for any I and $I_{\text{LRU}} \in \text{Worst}(I, C_N, \text{LRU})$, where I_{LRU} is overlap-free, we have that $\text{LRU}(I_{\text{LRU}}) \leq \text{FIFO}_W^{C_N}(I)$. Then, for any I, $\text{LRU}_W^{C_N}(I) \leq \text{FIFO}_W^{C_N}(I)$.*

Proof. Let $I_{\text{LRU}} \in \text{Worst}(I, C_N, \text{LRU})$. If I_{LRU} has an overlap, we show that by reordering while respecting C_N an overlap-free sequence with at least as many faults can be constructed.

Assume that I_{LRU} has an overlap and consider a first occurrence of a vertex u in I_{LRU} such that I_{LRU} contains the pattern $\langle \ldots, u, v^1, u, B, w, v^2, \ldots \rangle$, where u, v, and w are consecutive vertices on C_N. The superscripts on v are just for reference, i.e., v^1 and v^2 are the same vertex.

We define $I' = \langle \ldots, u, v^1, w, B^R, u, v^2, \ldots \rangle$, where B^R denotes the walk B, reversed. Clearly, I' respects C_N. We now argue that I' incurs no more faults than I_{LRU}. Clearly, there is a turn at v^1 in I_{LRU}. If there is also a turn at v^2, then we have effectively just removed two turns. Then Proposition 1 implies that I_{LRU} cannot be a worst ordering. Thus, we can assume there is no turn at v^2.

In the transformation, we are removing the turn at v^1 and introducing one at v^2. Thus, since in the sequence I_{LRU} all u-walks between turns contained at least $k-1$ vertices, this is still the case after the transformation in I', except possibly for the u-walk from the newly created turn at v^2 to the next turn in the sequence. Let x denote such a next turn.

If the u-walk between v and x has at least $k-1$ vertices, then the transformed sequence has the same number of turns, all u-walks between turns contain at least $k-1$ vertices, and therefore I_{LRU} and I' have the same number of hits (and faults). In addition, the state of the caches after treating I_{LRU} up to x and I' up to x are the same.

If that u-walk contains fewer than $k-1$ vertices, we consider the next turn y after x. Since there are at least $k-1$ vertices in between x and y, we must pass v on the way to y.

Thus, consider $I_{\mathrm{LRU}} = \langle \ldots, u, v^1, u, B, w, v^2, B_1, x, B_2, v^3, B_3, y, \ldots \rangle$, having turns at v^1, x, and y, versus $I' = \langle \ldots, u, v^1, w, B^R, u, v^2, B_1, x, B_2, v^3, B_3, y, \ldots \rangle$, where there are turns at v^2, x, and y.

Comparing $\langle \ldots, u, v^1, u, B, w, v^2 \rangle$ with $\langle \ldots, u, v^1, w, B^R, u, v^2 \rangle$, both of them have least $k-1$ vertices on any u-walk between two turns, and the latter has one fewer turns. Thus, by Proposition 1, it has $k-1$ fewer hits.

By assumption, B_1 has fewer than $k-1$ vertices. Thus, comparing I_{LRU} and I' up to and including x, I' has at least as many faults.

In I_{LRU}, $\langle B_2, v^3 \rangle$ must all be hits, so up to and including v^3, I' has at least as many faults.

Since the u-walk leading to v^1 in I' contains at least $k-1$ vertices (not including v^1), and since the u-walk going from v^2 to y goes in the same direction, the requests in $\langle B_3, y \rangle$ must all be faults in I'.

Thus, we have shown that there are at least as many faults in I' as in I_{LRU}. In addition, the state of the caches after treating I_{LRU} up to y and I' up to y are the same.

With the transformation above, we do not incur more faults, and any first occurrence of a vertex u initiating an overlap pattern has been moved further towards the end of the sequence. Thus, we can apply this transformation technique repeatedly until no more such patterns exist. □

We have now established the following property:

Property 4. We may assume that a worst ordering is overlap-free.

Now we have all the necessary tools to prove the theorem of this section.

Theorem 2. *For all I respecting the cycle C_N, $\mathrm{LRU}_W^{C_N}(I) \leq \mathrm{FIFO}_W^{C_N}(I)$.*

Proof. We may assume Properties 1–4.

Consider any I and $I_{\mathrm{LRU}} \in \mathrm{Worst}(I, C_N, \mathrm{LRU})$. If there are no turns at all in I_{LRU}, both FIFO and LRU will fault on every request. If there is only one turn, FIFO will clearly fault as often as LRU on I_{LRU}, since we may assume that there is no overlap.

So, consider the first two turns v and v'. By Property 4, we cannot have the pattern $\langle \ldots, u, v, u, B, w, v, \ldots \rangle$. Thus, after the first turn, the edge from w to v can never be followed again. This holds symmetrically for v', which is a turn in the other direction. Thus, once the request sequence enters the arc between v and v', it can never leave it again. We refer to this arc as the *gap*. To be precise, since we are on a cycle, the gap is the arc that at the two ends has the neighbor vertices of v and v' from which edges to v and v', respectively, cannot be followed again, and such that v and v' are not part of the arc.

Assume without loss of generality that, after the first turn, if the request sequence enters the gap between v and v', then it does so coming from v'. After the first turn at v, the requests can be assumed to be on the path access graph P_N instead of the cycle C_N, where the access graph P_N starts with v, continues in the direction of the turn at v, and ends at the neighbor of v in the gap. In fact, we can assume that we are working on the access graph P_N from $k-1$ requests before the first turn at v, since all u-walks can be assumed to have at least that length. Let r_i be that request. Since there are no turns before v, starting with r_i, LRU and FIFO function as they would starting with an empty cache.

We divide $I_{\mathrm{LRU}} = \langle r_1, r_2, \ldots, r_{|I_{\mathrm{LRU}}|}\rangle$ up into the sequences $\langle r_1, r_2, \ldots, r_{i-1}\rangle$ and $\langle r_i, \ldots, r_{|I_{\mathrm{LRU}}|}\rangle$. The former is a u-walk, where LRU and FIFO both fault on every request, and the latter can be considered a request sequence on a path access graph as explained above, and the conclusion follows from Theorem 1. □

5 Separation on a Path of Length $k+1$

In the last sections, we showed that LRU was at least as good as FIFO on any path graph or cycle graph. Now we show that LRU is strictly better if these graphs contain paths of length at least $k+1$. We exhibit a family of sequences $\{I_n\}_{n\geq 1}$ such that $\mathrm{FIFO}_W^{P_N}(I_n) \geq \left(\frac{k+1}{2}\right) \cdot \mathrm{LRU}_W^{P_N}(I_n) + b$, for some fixed constant b, on path graphs P_N with $N \geq k+1$. Only $k+1$ different pages are requested in I_n. The same family of sequences is also used to show that FWF is worse than either LRU or FIFO. We number the vertices of the path graph P_N in order from 1 through N.

Theorem 3. *There exists a family of sequences, $I_n = \langle 1, \ldots, k, k+1, k, \ldots, 2\rangle^n$, respecting the access graph P_N, and a constant b such that the following holds:*

$$\lim_{n\to\infty} \mathrm{LRU}(I_n) = \infty, \text{ and for all } I_n,\ \mathrm{FIFO}_W^{P_N}(I_n) \geq \left(\frac{k+1}{2}\right) \cdot \mathrm{LRU}_W^{P_N}(I_n) + b.$$

We now have tight upper and lower bounds on the relative worst order ratio of FIFO to LRU on paths.

Theorem 4. *For any access graph G, if $\mathrm{WR}_{\mathrm{FIFO,LRU}}^{G} \geq 1$, then $\mathrm{WR}_{\mathrm{FIFO,LRU}}^{G} \leq \frac{k+1}{2}$. Thus, if $N \geq k+1$, then $\mathrm{WR}_{\mathrm{FIFO,LRU}}^{P_N} = \frac{k+1}{2}$.*

It was shown in [7] that for a complete graph, the relative worst order ratio of FWF to FIFO is exactly $\frac{2k}{k+1}$. This is also a lower bound for any graph containing P_{k+1}, but it is still open whether or not equality occurs in all sparser graphs.

Theorem 5. *For any access graph G which has a path of length at least $k+1$,*

$$\mathrm{WR}_{\mathrm{FWF,FIFO}}^{G} \geq \frac{2k}{k+1} \text{ and } \mathrm{WR}_{\mathrm{FWF,LRU}}^{G} = k.$$

6 Incomparability

In this section, we show that on some general classes of access graphs, LRU and FIFO are incomparable. We consider the cyclic access graph defined by the edge set $\{(1,2), (2,3), (3,4), (4,5), (5,1)\}$ and the request sequence $I_1 = \langle 1, 5, 1, 2, 3, 4, 5, 1, 2, 1\rangle$ processed using a cache of size 4.

Lemma 5. *On any reordering of* I_1 *starting with* 1, LRU *incurs at least* 8 *faults and* FIFO *incurs at most* 7 *faults.*

Proof. It is trivial to check that LRU incurs 8 faults on I_1.

For FIFO, it is easy to check in the following that reorderings with repeated requests do not lead to more faults by FIFO. The reorderings of I_1 either have a prefix of the type $\{\langle 1, i, 1\rangle \mid i \in \{2, 5\}\}$ or $\{\langle 1, i, j\rangle \mid i \neq j \neq 1\}$. For the latter, examples being $\langle 1, 2, 3\rangle$ and $\langle 1, 5, 4\rangle$, the subsequence following the prefix contains 4 distinct pages. Since FIFO is conservative, it can incur at most 4 faults on that part after the prefix, bringing the total fault count up to at most 7.

The first four distinct page requests will always incur 4 faults, but for reorderings with the prefix $\{\langle 1, i, 1\rangle \mid i \in \{2, 5\}\}$, some pages are repeated within the first four requests. If the extended prefix is $\langle 1, i, 1, i\rangle$ for $i \in \{2, 5\}$, then the rest of the sequence still contains 4 distinct pages and again can add at most 4 faults to the previous 2, bringing the total up to at most 6. The only remaining case is a prefix of the form $\langle 1, i, 1, j\rangle$ where $i, j \in \{2, 5\}$, $i \neq j$. Here, there are 3 faults on the prefix. We divide the analysis of the rest of the sequence up into two cases depending on the next request following j:

For the first case, if the next request is 1, the extended prefix is $\langle 1, i, 1, j, 1\rangle$. However, then the next request to a page other than 1 is either to i or j and therefore not a fault. In addition, either there are no more i's or no more j's in the remaining part of the sequence, and again FIFO can then fault at most 4 times on this sequence with only 4 distinct pages.

For the second case, if the next request is $k \in \{3, 4\}$, then visiting $l \in \{4, 5 \,|\, l \neq k\}$ before the next j will give a prefix $\langle 1, i, 1, j, k, l\rangle$ with 5 faults, and the suffix must be $\langle i, 1, j, 1\rangle$ or $\langle i, 1, 1, j\rangle$, adding only one more fault. This gives 6 faults in total. If j is requested before l, the only possibilities are $\langle 1, 2, 1, 5, 4, 5, 1, 1, 2, 3\rangle$ and $\langle 1, 5, 1, 2, 3, 2, 1, 1, 5, 4\rangle$. In total, this gives only 5 faults. □

Note that the result above does not contradict our result about cycles. As predicted by that result, one of the worst orderings for LRU and FIFO would be $\langle 2, 1, 5, 4, 3, 2, 1, 5, 1, 1\rangle$, incurring 8 faults for both algorithms.

Using the cycle graph on which we processed I_1, we now construct a larger graph using "copies" of this graph as follows. For $2 \leq i \leq n$, we define I_i as a structural copy of I_1, i.e, we use new page names, but with the same relative order as in I_1 (like putting a "dash" on all pages in I_1). All these copies have their own set of pages such that no request in I_i appears in I_j for $i \neq j$. Just as I_1 implies a cycle graph that we denote X_1, so do each of these sequences and we let X_i denote the graph implied by I_i. Let $X_{i,k}$ denote the kth vertex in the ith copy and $I_{j,k}$ denote the kth request in the jth copy. To be precise, we define $I_i = \langle X_{i,1}, X_{i,5}, X_{i,1}, X_{i,2}, X_{i,3}, X_{i,4}, X_{i,5}, X_{i,1}, X_{i,2}, X_{i,1}\rangle$.

We define a graph $\mathcal{G}_n$ with a vertex set containing all $X_{i,k}$ and n additional vertices $u_1, u_2, \ldots, u_n$. Its edges are all the edges from the graphs X_i, $1 \leq i \leq n$, together with edges $(X_{i,1}, u_i)$ and $(u_i, X_{i+1,1})$ for all i, $1 \leq i \leq n-1$, plus the edge $(X_{n,1}, u_n)$.

Thus, $\mathcal{G}_n$ can be described as a chain of cycles, where each two neighboring cycles are separated by a single vertex. Clearly, the sequence $\mathcal{I}_n = \langle I_1, u_1, I_2, u_2, I_3, u_3, \ldots, I_n, u_n \rangle$ respects the access graph $\mathcal{G}_n$.

Theorem 6. *For the infinite family of sequences $\{\mathcal{I}_n\}$ respecting the access graph $\mathcal{G}_n$, the following two conditions hold:*

- $\lim_{n \to \infty} \mathrm{FIFO}(\mathcal{I}_n) = \infty$.
- *for all $\mathcal{I}_n$, $\mathrm{LRU}_W^{\mathcal{G}_n}(\mathcal{I}_n) \geq \frac{9}{8} \cdot \mathrm{FIFO}_W^{\mathcal{G}_n}(\mathcal{I}_n)$.*

Thus, although LRU is strictly better than FIFO on paths and cycles, FIFO is strictly better than LRU on the family of sequences, $\{\mathcal{I}_n\}$, respecting the family of graphs, $\{\mathcal{G}_n\}$. Note that $J_r = \langle X_{1,1}, X_{1,2}, X_{1,3}, X_{1,4}, X_{1,5}, X_{1,4}, X_{1,3}, X_{1,2} \rangle^r$, which only uses the first cycle of $\mathcal{G}_n$, trivially respects $\mathcal{G}_n$ for any r. By Theorem 3 for $k = 4$, $\mathrm{FIFO}_W^{\mathcal{G}_n}(J_r) \geq \left(\frac{k+1}{2}\right) \mathrm{LRU}_W^{\mathcal{G}_n}(J_r) - (k-1)$.

Thus, on the family $\{J_r\}$, LRU is better than FIFO. Combining with Theorem 6, we get:

Theorem 7. LRU *and* FIFO *are incomparable on the family of graphs $\{\mathcal{G}_n\}$, according to relative worst order analysis.*

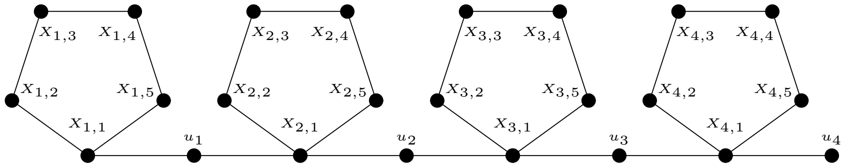


Fig. 1. The graph $\mathcal{G}_n$ for $n = 4$

7 Open Problems

We have determined that according to relative worst order analysis, LRU is better than FIFO on paths and cycles. On some classes of general access graphs, the two algorithms are incomparable. It would be interesting to get closer to determining exact access graphs classes characterizing relationships between the two algorithms. We believe that the results for paths and cycles will form fundamental building blocks in an attack on this problem. The most obvious class of access graphs to study next is trees. LRU can clearly do better than FIFO on any tree containing a path of length $k + 1$. We conjecture that LRU does at least as well as FIFO on any tree. One difficulty in establishing a proof of this is that for trees, as opposed to the cases of paths and cycles, there exist worst order sequences for LRU for which FIFO performs better than LRU.

For general access graphs, when showing that FIFO can do better than LRU, we used a family of access graphs, the size of which grew with the length of the input sequence. It would be interesting to know if this is necessary, or if such a separation result can be established on a single access graph of bounded size.

References

1. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inform. Proc. Letters 56, 135–139 (1995)
2. Albers, S., Westbrook, J.: Self-Organizing Data Structures. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms 1996. LNCS, vol. 1442, pp. 13–51. Springer, Heidelberg (1998)
3. Bentley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Comm. ACM 28, 404–411 (1985)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
5. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. Journal of Computer and System Sciences 50(2), 244–258 (1995)
6. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. ACM Transactions on Algorithms 3(2), article no. 22 (2007)
7. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst order ratio applied to paging. Journal of Computer and System Sciences 73(5), 818–843 (2007)
8. Boyar, J., Gupta, S., Larsen, K.S.: Access graphs results for LRU versus FIFO under relative worst order analysis, arXiv:1204.4047v1 [cs.DS] (2012)
9. Chrobak, M., Noga, J.: LRU is better than FIFO. Algorithmica 23(2), 180–185 (1999)
10. Denning, P.J.: The working set model for program behaviour. Comm. ACM 11(5), 323–333 (1968)
11. Denning, P.J.: Working sets past and present. IEEE Transactions on Software Engineering 6(1), 64–84 (1980)
12. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News 36(3), 67–81 (2005)
13. Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: List Factoring and Relative Worst Order Analysis. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 118–129. Springer, Heidelberg (2011)
14. Fiat, A., Karlin, A.R.: Randomized and multipointer paging with locality of reference. In: 27th Annual ACM Symposium on Theory of Computing, pp. 626–634 (1995)
15. Fiat, A., Mendel, M.: Truly online paging with locality of reference. In: 38th Annual Symposium on Foundations of Computer Science, pp. 326–335 (1997), extended version: CoRR, abs/cs/0601127 (2006)
16. Fiat, A., Rosen, Z.: Experimental studies of access graph based heuristics: Beating the LRU standard? In: 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 63–72 (1997)
17. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Systems Tech. Journal 45(9), 1563–1581 (1966)
18. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM J. Comput. 25(3), 477–497 (1996)
19. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. Journal of the ACM 47(4), 617–643 (2000)
20. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 79–119 (1988)
21. Karlin, A.R., Phillips, S.J., Raghavan, P.: Markov paging. SIAM J. Comput. 30(3), 906–922 (2000)
22. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Comm. ACM 28(2), 202–208 (1985)

Competitive Analysis of Maintaining Frequent Items of a Stream[*]

Yiannis Giannakopoulos[1] and Elias Koutsoupias[2]

[1] Department of Informatics
University of Athens
ygiannak@di.uoa.gr

[2] Department of Informatics
University of Athens
elias@di.uoa.gr

Abstract. We study the well-known frequent items problem in data streams from a competitive analysis point of view. We consider the standard worst-case input model, as well as a weaker distributional adversarial setting. We are primarily interested in the single-slot memory case and for both models we give (asymptotically) tight bounds of $\Theta(\sqrt{N})$ and $\Theta(\sqrt[3]{N})$ respectively, achieved by very simple and natural algorithms, where N is the stream's length. We also provide lower bounds, for both models, in the more general case of arbitrary memory sizes of $k \geq 1$.

1 Introduction

The *frequent items* problem [8] is one of the most well-studied ones in the area of data streams [17,1,13,12,4]. Informally, the problem is that of observing a stream (sequence) of values and trying to discover those that appear most frequently. Many applications in packet routing, telecommunication logging and tracking keyword queries in search machines are critically based upon such routines.

More formally, in the most basic version of the classic frequent items problem, we are given a stream $a_1, a_2, \dots, a_N$ of items from some universe, as well as a frequency threshold ϕ, $0 < \phi < 1$, and we are asked to find and/or maintain all items that occur more than ϕN times throughout the stream. For real-life applications there are some restricting assumptions the algorithms need to respect: the size N of the stream, as well as the rate at which the items arrive, far exceed the computational capabilities of our devices. Consequently, we usually require streaming algorithms to use $O(\mathit{polylog}(N))$ memory and allow approximate solutions within a factor of ε (additive, with respect to ϕ), since exact solutions would require linear space [8], a totally unrealistic option in some domains. Furthermore, we are usually interested in algorithms that make as few passes as possible over the input stream and, in particular, *single-pass* algorithms that process the input stream in an online way, i.e. each item sequentially, making decisions on-the-fly.

[*] This work has been partially supported by the ESF-NSRF research program Thales (ALGONOW). The first author acknowledges the support of Propondis Foundation.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 340–351, 2012.

So, traditionally data stream problems have been essentially approached as space complexity optimization problems: given an input stream and an approximation guarantee of ϵ, we try to minimize the memory used (see, e.g., the seminal work of [2]). However, despite their intrinsic *online* nature, these problems have not been studied within the predominant framework for studying online problems, i.e. that of competitive analysis [6].

Only recently, Becchetti and Koutsoupias [5] did that, using competitive analysis to study another important streaming problem, namely that of maintaining the maximum value within a sliding window [9]. Following a similar approach, we formulate an online version of the classic frequent items problem: given an input stream and a memory of at most k slots, try to optimize the online algorithm's performance with respect to that of an optimal offline algorithm that knows the entire input stream in advance. This is, in a way, the inverse of the traditional space complexity optimization objective mentioned above. We also use a similar aggregate-through-time perspective (instead of an unrealistic optimization-at-every-step requirement) upon defining our objective function (see section 1.1). As argued in [5], the competitive analysis approach combined with an aggregate objective, seem more appropriate for economic applications as well as a decision-making under uncertainty framework.

1.1 Setting

We are observing a stream $A = a_1, a_2, \ldots, a_N$ of N elements drawn from some universe $\mathcal{U}$, in a sequential, *online* fashion. It is natural to assume that $|\mathcal{U}| \gg N$. We consider algorithms that, at every time step $t = 1, 2, \ldots, N$, maintain in memory a set $S_t(A) \subseteq \{a_1, a_2, \ldots, a_t\}$ of at most k items from the part of the input stream A observed so far[1]. Furthermore, we assume that the only way in which our algorithms can update these memory sets throughout the execution, is to make an *irrevocable* decision, at every time point t, of whether or not to store the newly arrived element a_t in memory, i.e. $S_t \subseteq S_{t-1} \cup \{a_t\}$. An *online* algorithm can base its decision just[2] on the knowledge of its current memory state S_{t-1} and the current time point t, while an offline algorithm can have access to the entire input stream A. Notice that, since $|S_t| \leq k$, if at some point t we want to store a new element $a_t \notin S_{t-1}$, then we may need to discard some previously stored item $a_j \in S_{t-1}$, $j < t$ and then $S_t \subseteq \{a_t\} \cup S_{t-1} \setminus \{a_j\}$. For the special case of $k = 1$, which will be our main concern in this paper for the most part (we will consider general memory sizes of $k \geq 1$ again in section 4) we will denote by s_t the unique item in the algorithm's memory at time point t, i.e. $S_t = \{s_t\}$.

Given an input stream A and an element $a \in A$ we define its *frequency* as $f^A(a) = \frac{n^A(a)}{N}$, where $n_A(a) = |\{i \mid a_i = a\}|$ is the number of instances of a

[1] To keep notation light, we will simply use S_t instead of $S_t(A)$ whenever it is clear to which input stream we are referring to.

[2] In a different online setting, we could have also assumed that the online algorithm has complete knowledge of the past. This, though, seems as a rather unrealistic assumption to make, especially in a streaming setting.

in the stream[3]. Intuitively, we want our algorithms, at every time, to maintain the most frequent items possible. We formalize this, by defining the *aggregate frequency* objective as the sum, across the entire execution, of the frequencies of all *distinct* items in memory, i.e. $\sum_{t=1}^{N}\sum_{a\in S_t} f(a)$. Notice here a fine point: we treat S_t as a set and *not* as a multi-set, i.e. multiple occurrences of the same element in memory can only contribute *once* towards our objective. We measure an online algorithm's performance on a given input A by comparing its total gain (i.e. the value of the aggregate frequency objective on stream A) to that of an offline algorithm that knows the entire input stream A in advance. The *competitive ratio* of the online algorithm is the maximum value of this ratio among all possible inputs,

$$\max_A \frac{\sum_{t=1}^{N}\sum_{a\in S'_t} f^A(a)}{\sum_{t=1}^{N}\sum_{a\in S_t} f^A(a)},$$

where S_t, S'_t are the memory sets of the online and optimal offline algorithm, respectively. The competitive ratio for our online frequent items problem, is the best (minimum) competitive ratio we can achieve over all online algorithms A.

Finally, whenever we deal with randomized algorithms in this paper, we are always silently assuming the standard, oblivious adversary [11,6] model, i.e. the adversary decides an input A knowing the online algorithm but not the actual results of its coin tosses.

1.2 Organization of the Paper and Results

In this paper we are mostly interested in the special case of single-slot memories ($k = 1$). That is the case for Sections 2 and 3. We do not deal with general memory sizes of $k \geq 1$ until Section 4.

In Section 2, we prove that the competitive ratio of the online frequent items problem is $\Theta(\sqrt{N})$, by providing a lower bound proof of $\frac{1}{3}\sqrt{N}$ and showing that the most simple algorithm that myopically accepts every element that arrives achieves an (asymptotically) tight $\sqrt{N}$ competitive ratio. Furthermore, in Section 2.1, we show that the well known Majority algorithm for the classical frequent items problem performs very poorly from a competitive analysis point of view, since it has (asymptotically) the worst possible competitive ratio an online algorithm can demonstrate, namely $\Theta(N)$.

Also, we consider weaker adversarial inputs and in particular the case of the input stream being generated i.i.d. from a probability distribution, known only to the adversary. In section 3 we show how a very simple and natural algorithm, called Eager, that essentially waits until it sees some element appearing twice, achieves a competitive ratio of $O(\sqrt[3]{N})$, asymptotically matching a lower bound of $\Omega(\sqrt[3]{N})$ again providing a tight competitive ratio (of $\Theta(\sqrt[3]{N})$) for the case of $k = 1$.

Next, in Section 4 we deal with the more general case of arbitrary memory sizes of $k \geq 1$, for which we give lower bounds of $\frac{1}{3}\sqrt{N/k}$ and $\Omega(\sqrt[3]{N/k})$, respectively,

[3] Again, we will drop the superscript A whenever this causes no confusion.

for both the worst-case and weaker distributional adversarial models. In fact, the lower bounds of previous Sections 2 and 3 are derived by simply setting $k = 1$ at these more general bounds.

Finally, in section 5 we provide an extensive discussion regarding interesting possible extensions to our model and open problems.

2 Worst-Case Bounds

As the following Theorem 1 demonstrates, one can not hope for online algorithms with bounded competitive ratios. In particular:

Theorem 1. *No (randomized) algorithm for the online frequent items problem can have a competitive ratio better than* $\frac{1}{3}\sqrt{N} - o(1)$*.*

Proof. The proof can be found in section 4 where we prove the more general Theorem 13 for arbitrary memory sizes of $k \geq 1$.

The lower bound of Theorem 1 is (asymptotically) tight and achieved by the most simple deterministic algorithm:

Definition 2 (Algorithm Naive**).** *The deterministic* Naive *algorithm accepts every element as it arrives. Formally,* $s_t = a_t$ *for all* $t = 1, 2, \ldots, N$*.*

Theorem 3. *The* Naive *algorithm is* $\sqrt{N}$*-competitive.*

Proof. Fix a worst-case input stream A for the Naive algorithm, let α be an element of the stream having the highest frequency, i.e. $\alpha = \text{argmax}_{a \in A} f(a)$ and let $f = f(\alpha)$. First, notice that the optimal offline gain cannot exceed Nf (by simply giving to it α from the very start).

Next, we provide two lower bounds on the online gain. First, we have a trivial lower bound of $N\frac{1}{N} = 1$, since every element stored at any time t in our memory has a frequency of at least $\frac{1}{N}$. Next, since every element $a \in A$ is being accepted as it arrives, we have it in our memory for at least $Nf(a)$ times for a gain of $f(a)$ per time, so the online gain is at least $Nf \cdot f = Nf^2$.

Consider two cases: If $f < \frac{1}{\sqrt{N}}$, then we use the first bound on the online gain to get a competitive ratio of at most $\frac{Nf}{1} < \frac{N}{\sqrt{N}} = \sqrt{N}$ and at the complimentary case of $f \geq \frac{1}{\sqrt{N}}$ we use the other bound to get, again, a competitive ratio of $\frac{Nf}{Nf^2} = \frac{1}{f} \leq \sqrt{N}$.

2.1 The Classic Majority Algorithm

As was discussed in our introduction, if we demand single-pass algorithms then the traditional data streaming setting is essentially an online setting and so, some algorithms for the classic frequent items problem are valid algorithms for our online version too. This is true for the *counter-based* algorithms (see [8]) and in particular for the well-known Majority algorithm [7], for the case of $k = 1$, and its generalizations (see, e.g., the Misra-Gries algorithm [14]) for the case of $k > 1$.

The Majority algorithm keeps a single item in memory, along with a counter, initialized to 0. If the next value that arrives is the same as the one currently stored, then increase the counter by $+1$, otherwise decrease it by -1. Whenever the counter reaches 0, flush the memory and accept the next item to arrive, increasing the counter to 1. This simple and clever algorithm manages to solve the 1/2-approximation classic frequent items problem (i.e. if there is a majority element, appearing more than half the time, it is then in the algorithm's memory at the end of the execution) using just a single memory slot. However, as Theorem 4 demonstrates, from a competitive analysis point of view, its performance is disappointingly poor. The intuition behind this failure, which is the reason why other classic algorithms would also not perform efficiently for the online problem, is that the ε-approximate deviation per step has a drastic effect when aggregated across the entire execution as $N \to \infty$.

It is a simple observation that no online algorithm for the frequent items problem can perform worse than N with respect to the competitive ratio: at every given step, the offline gain can not exceed 1 and the online gain can not be less than $1/N$, since for every element α in the stream, $\frac{1}{N} \leq f(\alpha) \leq 1$.

Theorem 4. *The Majority algorithm (asymptotically) achieves the worst possible competitive ratio of $\Theta(N)$.*

Proof. Fix a stream length N, N being even, and pick $\frac{N}{2}+1$ distinct elements $\alpha, \alpha_1, \alpha_2, \dots, \alpha_{N/2} \in \mathcal{U}$. Give as input the stream

$$\alpha_1, \alpha, \alpha_2, \alpha, \dots, \alpha_{N/2}, \alpha.$$

On this input, Majority is forced to change every two steps, with $s_{2i-1} = s_{2i} = \alpha_i$ for all $i = 1, 2, \dots, \frac{N}{2}$, never having the "desired" element α in memory, resulting to a disappointing gain of $N \cdot \frac{1}{N} = 1$, while the offline strategy that puts element α in memory as soon as it arrives at time point $t = 2$ and keeps it until the end achieves a gain of $\frac{1+(N-1)\frac{N}{2}}{N}$, giving a competitive ratio of at least $\frac{N}{2} - \frac{1}{2}$.

3 Weaker Adversarial Models

Here we consider a distributional model where the adversary, instead of explicitly selecting the input sequence stream A, now just decides on a particular probability distribution, unknown to the online algorithm, which we sample to construct the input.

Definition 5 (Distributional adversarial model). *In the distributional model the adversary decides the stream's length N and picks a probability distribution $\mathcal{D}$ on the universe of elements $\mathcal{U}$. The input stream A is generated by drawing N observations i.i.d. from $\mathcal{D}$. $\mathcal{D}$ is not known to the online algorithms.*

It is interesting to notice that this type of adversary resembles the random order input model used in the classical secretary problems as well as recent online auctions settings (see, e.g.,[3,10]). A random order streaming model was first proposed in the seminal paper of Munro and Paterson [16].

Notation. In this section, we will use the following notation: Let N be the length of the stream, $\mathcal{D}$ be the distribution of the input, having support $D = \{\alpha_1, \alpha_2, \alpha_3, \dots\}$ with probability p_i corresponding to element α_i. We set $q = \sum_{\alpha_i \in D} p_i^2$ and without loss of generality, assume that $p_1 \geq p_2 \geq \dots$. Since the items of the stream are selected i.i.d. from $\mathcal{D}$, the expected frequency of element α_i at the resulting stream is simply $f(\alpha_i) = p_i$, for all $\alpha_i \in D$.

We first provide a lower bound. Although obviously better than that of the worst-case model of section 2, it still remains unbounded with respect to the stream's length N:

Theorem 6. *The competitive ratio of the online frequent items problem for the adversarial model is $\Omega(\sqrt[3]{N})$.*

Proof. As with Theorem 1, the proof can be found in section 4 for the more general Theorem 14 for arbitrary memory sizes of $k \geq 1$.

Now we turn our attention to upper bounds, and in search for an online algorithm that, used in our current model of distributional inputs, will break the $\Omega(\sqrt{N})$ bound for the worst-case model of Section 2, the first question we should deal with is whether a trivial algorithm such as the Naive algorithm of Definition 2, which does not even take into consideration its memory set S_{t-1} before making a decision, can achieve this. The answer is negative:

Theorem 7. *Every (randomized) online algorithm that does not take into consideration its memory set, i.e., for all time points t, its probability of accepting incoming element a_t can depend only on the current absolute execution time t and not S_{t-1}, has a competitive ratio of $\Omega(\sqrt{N})$ in the distributional adversarial model.*

Proof. Fix a stream length N with N being an even, perfect square positive integer and let $m = \sqrt{N}$. Consider a probability distribution $\mathcal{D}$ with support $\{\alpha_0, \alpha_1, \dots, \alpha_{N-m}\} \subseteq \mathcal{U}$, for which $p_0 = m/N$ and $p_j = 1/N$ for all $j = 1, 2, \dots, N-m$. Let P_t denote the probability of accepting the newly arrived element a_t at time t and $Q_t = \Pr[s_t = \alpha_0]$, the probability of having element α_0 stored in memory at time t. Then

$$Q_{t+1} = Q_t \cdot [1 - (1 - p_0) P_{t+1}] + (1 - Q_t) \cdot p_0 P_{t+1} = P_{t+1} \left(\frac{m}{N} - Q_t \right) + Q_t,$$

which, by the fact that $Q_1 = m/N$ (since at the first step any online algorithm just puts item a_1 in memory), gives that $Q_t = \frac{m}{N}$ for all t. That means that the expected gain of our online algorithm is

$$\begin{aligned} \sum_{t=1}^{N} [Q_t f(\alpha_0) + (1 - Q_t) f(\alpha_1)] &= \sum_{t=1}^{N} \left[Q_t \frac{m}{N} + (1 - Q_t) \frac{1}{N} \right] \\ &= \frac{m^2}{N} - \frac{m}{N} + 1 = 2 - \frac{1}{\sqrt{N}}. \end{aligned}$$

The offline algorithm that waits until it sees the desired element α_0 and then stores it in memory forever has asymptotically an expected gain of at least $\frac{N}{2}f(\alpha_0) = \frac{N}{2}\frac{m}{N} = \frac{\sqrt{N}}{2}$, since the probability of having discovered α_0 until the middle of the stream is $1-(1-p_0)^{N/2} = 1-\left(1-\frac{1}{\sqrt{N}}\right)^{N/2} \to 1$, as $N \to \infty$.

Theorem 7 above tells us that we should consider only non-trivial algorithms that consult their memory before making a decision. The simplest of them is the following:

Definition 8 (Algorithm Eager**).** *The* Eager *algorithm accepts every element as it comes. If it finds the same element in two consecutive positions, then it keeps it until the end. Formally, let*

$$t^* = \min_{1\le t\le N-1}\{t \mid a_t = a_{t+1}\}$$

if that exists, otherwise $t^* = N$. Then Eager *is the algorithm with $s_t = a_t$ for all $t \le t^*$ and $s_t = a_{t^*}$ for all $t > t^*$.*

First, we need to bound the optimal offline gain:

Theorem 9. *The expected optimal offline gain of the distributional model is $O(\max\{N^{1/3}, p_1 N\})$.*

Proof. Let X_i be the random variable representing the number of appearances of element α_i at the resulting stream (generated i.i.d. from $\mathcal{D}$). Then, X_i follows a binomial distribution with parameters p_i, N. It is clear that the expected optimal offline gain cannot exceed neither $\max_i X_i$ ($\frac{\max_i X_i}{N}$ per step), nor N (1 per step). Also, let $M = \max\{N^{1/3}, p_1 N\}$, $I_1 = \left\{i \mid p_i N \ge N^{1/3}\right\}$ and $I_2 = \left\{i \mid p_i N < N^{1/3}\right\}$.

We will need the following lemma:

Lemma 10. *For all $i \in I_1$, $\Pr[X_i > 6p_i N] \le 2p_1 p_i$ and for all $i \in I_2$, $\Pr\left[X_i > 6N^{1/3}\right] \le 2p_i N^{-2/3}$.*

Proof. If $i \in I_1$, then $p_1 \ge p_i \ge N^{-2/3}$ and using a Chernoff bound (see [15, Theorem 4.4]) we get

$$\Pr[X_i > 6p_i N] \le 2^{-6p_i N} \le 2p_1 p_i,$$

since $\frac{2^{-6p_iN}}{p_i p_1} \le \frac{2^{-6p_iN}}{p_i^2} \le 2^{-6N^{1/3}}N^{4/3} \le 2$ for all N, the second inequality holding by substituting $p_i = N^{-2/3}$ at a monotonically decreasing function.

If $i \in I_2$, then $p_i < N^{-2/3}$ and

$$\begin{aligned}\Pr\left[X_i > 6N^{1/3}\right] &\le \sum_{k=6N^{1/3}}^{N}\binom{N}{k}p_i^k(1-p_i)^{N-k}\\ &\le p_i^{6N^{1/3}}\binom{N}{6N^{1/3}}{}_2F_1(6n^{1/3}-n,1;1+6n^{1/3};-\frac{p_i}{1-p_i}),\end{aligned}$$

where ${}_2F_1$ is the Gaussian hypergeometric function, maximized at $p_i = N^{-2/3}$ for a value of less than 2 (for all N), giving an upper bound, for the above probability, of $2p_i^{2N^{1/3}}\binom{N}{6N^{1/3}}$. So, we need to show that $p_i^{6N^{1/3}}\binom{N}{6N^{1/3}} \leq p_i N^{-2/3}$, i.e. it is enough to show that

$$N^{2/3} p_i^{6N^{1/3}-1}\binom{N}{6N^{1/3}} \leq N^{2/3}\left(N^{-2/3}\right)^{6N^{1/3}-1}\binom{N}{6N^{1/3}} \leq 1,$$

which holds for all N, concluding the proof of the lemma.

Now we are ready to bound the expected optimal offline gain:

$$\begin{aligned}\mathbf{E}[\max X_i] &\leq 6M \cdot \Pr\left[\max X_i \leq 6M\right] + N \cdot \Pr\left[\max X_i > 6M\right] \\ &\leq 6M + N\left(\sum_{i\in I_1}\Pr\left[X_i > 6M\right] + \sum_{i\in I_2}\Pr\left[X_i > 6M\right]\right) \\ &\leq 6M + N\left(\sum_{i\in I_1}\Pr\left[X_i > 6p_iN\right] + \sum_{i\in I_2}\Pr\left[X_i > 6N^{1/3}\right]\right),\end{aligned}$$

since $M \geq p_1 N \geq p_i N$ and $M \geq N^{1/3}$, and so

$$\mathbf{E}[\max X_i] \leq 6M + N\left(\sum_{i\in I_1} 2p_1p_i + \sum_{i\in I_2} 2N^{-2/3}p_i\right) \leq 6M + 2M\sum_i p_i = 8M.$$

Next, we turn our attention to analyzing the online gain of our algorithms:

Lemma 11. *The* Naive *algorithm (see Definition 2) has an expected gain of at least* $1 + q(N-1)$ *for the distributional model, where* $q = \sum p_i^2$.

Proof. Notice, that, since the Naive algorithm accepts every element to arrive, at every step it has element α_i stored in memory with probability p_i, for an expected (frequency) gain of $\frac{1+p_i(N-1)}{N}$ at this step, resulting to a total expected gain of $1 + q(N-1)$.

We are ready now to give our upper bound, asymptotically matching that of Theorem 6:

Theorem 12. *The randomized algorithm that runs* Naive *and* Eager *with an equal probability of* $\frac{1}{2}$ *is* $O(\sqrt[3]{N})$*–competitive for the distributional model.*

Proof. From Lemma 11 and Theorem 9 we only need to show that Eager is $O(N^{1/3})$–competitive for the case when $p_1 N > N^{1/3}$ and $\frac{p_1 N}{1+q(N-1)} = \omega(N^{1/3})$, which give the following necessary conditions:

$$p_1 > N^{-2/3} \quad \text{and} \quad q < \frac{p_1}{N^{1/3}} \ . \tag{1}$$

Let t^* be as in Definition 8. Then, since the stream is generated i.i.d. from $\mathcal{D}$, $\Pr\left[t^* \geq t\right] = \prod_{i=1}^{t-1}\Pr\left[a_i \neq a_{i+1}\right] = \prod_{i=1}^{t-1}\left(1 - \Pr\left[a_i = a_{i+1}\right]\right) =$

$\prod_{i=1}^{t-1}(1-\sum_i p_i p_i) = (1-q)^{t-1}$ for $t \le N-1$, and so the probability of Eager "discovering" α_1 at time $t+1$ is $(1-q)^{t-1}p_1^2$, for an expected gain of at least $\frac{2+(N-2)p_1}{N}(N-t+1) \ge p_1(N-t+1)$, resulting to a total expected gain of at least

$$\sum_{t=1}^{N-1}(1-q)^{(t-1)}p_1^2 \cdot p_1(N-t+1) = p_1^3\frac{(1-2q)(1-q)^{N-1}+q(N+1)-1}{q^2}$$

This, together with Theorem 9 and (1), gives a competitive ratio upper bound of

$$\frac{F(q,N)}{p_1^2} \quad \text{where} \quad F(q,n) = \frac{q^2 N}{(1-2q)(1-q)^{N-1}+q(N+1)-1} \ .$$

It is not difficult to see that $F(q,n)$ is an increasing function with respect to q, thus by (1) getting an upper bound of

$$\frac{F(p_1/N^{1/3},N)}{p_1^2} = \frac{N^{1/3}}{\left(1-\frac{2p_1}{n^{1/3}}\right)\left(1-\frac{p_1}{N^{1/3}}\right)^{N-1}+p_1\left(\frac{1}{N^{1/3}}+N^{2/3}\right)-1} \ .$$

The denominator in the above expression is an increasing function with respect to p_1 and so, by (1), it is minimized by setting $p_1 = N^{-2/3}$ for a value of $\left(1-\frac{2}{N}\right)\left(1-\frac{1}{N}\right)^{N-1}+\frac{1}{N} \to \frac{1}{e}$, as $N \to \infty$, giving the desired upper bound of $O(N^{1/3})$ on our competitive ratio.

4 Lower Bounds for Arbitrary Memory *k*

In this section we consider an arbitrary memory size of $k \ge 1$, generalizing the single-slot memory settings of the previous Sections 2 and 3 and we extend the lower bounds for both the worst-case and the weaker distributional adversarial models considered in these sections. Obviously, for $k=1$ our results here match those of Theorems 1 and 6, respectively.

Theorem 13. *No (randomized) algorithm for the online frequent items problem can have a competitive ratio better than* $\frac{1}{3}\sqrt{\frac{N}{k}}-o(1)$.

Proof. We use Yao's principle [18,6]. Fix a stream length N and set $m=\sqrt{\frac{N}{k}}$ (which we assume to be integer). Fill the stream's first $N-km$ positions, with $N-km$ distinct elements $a_1,\ldots,a_{N-km} \in \mathcal{U}$. The probabilistic input is constructed by selecting uniformly at random k distinct elements $a_{j_1},\ldots,a_{j_k}$ from the first km positions and using m copies of each to fill the remaining last km slots of the stream.

An offline algorithm picks an element α_{j_i} as soon as it appears, an event which occurs at step km at the latest, and keeps it until the end, resulting in a total gain of at least $\frac{(N-km)\cdot(m+1)}{N} = m - \frac{1}{m}$ for each of the k positions in memory.

By the uniform way in which our input is constructed, it is easy to see that the best online deterministic strategy is to pick the first k elements, keep them until position $N-km$ and then gradually replace them with the k distinct elements of the last km positions. For a given memory slot, the probability that the online algorithm will succeed to initially pick one of the high-frequency items is at most $\frac{k}{km}=1/m$. In this case the gain is at most $N\frac{m+1}{N}=m+1$. Otherwise, the gain is at at most $(N-km)\frac{1}{N}+km\frac{m+1}{N}=1+\frac{km^2}{N}=2$. Therefore the expected online gain per each memory slot is at most $(m+1)\frac{1}{m}+2(1-\frac{1}{m})\leq 3$. It follows that the ratio of the optimal gain over the online gain is at least $\frac{1}{3}(m-\frac{1}{m})=\frac{1}{3}\sqrt{\frac{N}{k}}-o(1)$.

Theorem 14. *The competitive ratio of the online frequent items problem for the adversarial model is $\Omega(\sqrt[3]{N/k})$.*

Proof. We use again Yao's principle and give a distributional input $\mathcal{D}$ with k elements $\alpha_1,\alpha_2,\ldots,\alpha_k$ (not known to the online algorithm) having a probability of $p=(kN^2)^{-1/3}$ and all other elements in $\mathcal{U}$ being assigned arbitrarily small probabilities of $\varepsilon\to 0$ (we are assuming a very large universe $|\mathcal{U}|\gg k$). Notice that this is a valid probability distribution, since $kp<1$. By this construction, it is safe to assume that no element except $\alpha_1,\alpha_2,\ldots,\alpha_k$, appears more than once in the resulting stream $a_1,a_2,\ldots,a_N$, since such an event occurs with very small, negligible probability. So, when an online algorithm observes an item that it is already in its memory (i.e. $a_t\in S_{t-1}$), it knows for sure that this is one of the "good" elements $\alpha_1,\alpha_2,\ldots,\alpha_k$ and, furthermore, until such a "marking" occurs the online algorithm can have no information on the identities of these desired elements, meaning that, until then, the probability of having α_i residing in a particular memory slot is at most p, for all $i=1,2,\ldots,k$ and all memory slots. Thus, the probability of discovering α_i, $i=1,2,\ldots,k$, at some time point t can not exceed $kp\cdot p$ (at most k slots are still "free" at time t and each is holding α_i with a probability of at most p) an event giving an expected gain of at most Np (assume that all discovered elements can be stored, that we are receiving gain from all possible multiple copies of the same element and that this gain is taken from the start $t=1$). Also, if we don't discover any "good" element throughout the execution, we can trivially get an expected gain of at most $kp\cdot p+(1-kp)\varepsilon$ per step and per slot. Summing up, the expected total gain of every online deterministic algorithm is at most

$$Nk\left[Nkp^3+kp^2+(1-kp)\varepsilon\right]\leq 2k+Nk\varepsilon\approx 2k,$$

since $p=(kN^2)^{-1/3}$, $k\leq N$ and $\varepsilon\to\infty$.

Finally, consider the offline algorithm that waits until it sees element α_i for the first time and then stores it in the i-th memory slot forever (for an expected gain of at least p per step). Notice that the probability that an element α_i has not been stored until halfway the execution is $(1-p)^{N/2}\leq 1/\sqrt{e}$ (since $p\geq 1/N$), so the expected offline gain is at least $k\left(1-\frac{1}{\sqrt{e}}\right)\frac{N}{2}p=\frac{(1-e^{-1/2})}{2}N^{1/3}k^{2/3}$, giving the desired competitive ratio.

5 Extensions and Future Directions

The most obvious open problem based on our work is that of closing the gap, in both adversarial models, between the upper and lower bounds of the competitive ratios with respect to the memory size parameter k in the general case of arbitrary memory sizes of $k > 1$.

Throughout this paper we have used an *absolute* frequency notion, i.e. we take an element's a count $n^A(a)$ (number of appearances) with respect to the entire stream A. Alternatively, one can define an *ephemeral* frequency $f_t^A(a)$, dependent on the current time point t and refer to element's a count up to that point, i.e.

$$f_t^A(a) = \frac{|\{i \leq t \mid a_i = a\}|}{t}. \tag{2}$$

Notice that, as expected, $f(a) = f_N(a)$. Our aggregate frequency objective can be naturally extended to $\sum_{t=1}^{N} \sum_{a \in S_t} f_t(a)$.

Intuitively, the competitive ratios for ephemeral frequencies would essentially not change for the distributional adversarial model, since the expected ephemeral frequency of an element at a given time t would be equal to its expected absolute frequency (resulting to equal expected gains per step), while in the worst-case model one should expect better performance: the "nemesis" inputs similar to that of the lower bound proof of Theorem 1 no longer apply in such a disastrous way. Further comparison of the two models is left for the journal version of this work.

Another interesting direction that the adoption of ephemeral frequencies gives us, is that of considering a measure of importance upon our time horizon, e.g. in many scenarios we are more interested in the recent past rather than remote observations. This can be modeled by using a non-decreasing, non-negative real valued *aging sequence* $q_0, q_1, q_2, \ldots$ in order to, for a given time point t, assign to past element a_{t-i} a weight of q_i, where $i = 0, 1, \ldots, t-1$. From this point of view, we need to generalize our (ephemeral) frequency definition in (2) to

$$f_t^A(a) = \frac{\sum_{i:\ a_{t-i}=a} q_i}{\sum_{i=0}^{t-1} q_i},$$

to capture this notion of time decay. Examples of such aging sequences may include an exponentially decreasing time model, where $q_i = e^{-\lambda \cdot i}$ for some real constant $\lambda > 0$, or a sliding window model of window size w, where $q_i = 1$ if $i \leq w - 1$ and $q_i = 0$ otherwise. Notice that the simple streaming model with no time-decaying which we have considered in this paper corresponds to the case of $q_i = 1$ for all i.

Finally, as stated in the introduction, to our knowledge this is just the second work that studies some data streaming problem in a competitive analysis framework, after only the recent Aggregate-Max paper of Becchetti and Koutsoupias [5]. We think that a lot of interesting work can be further done in this direction and that the intrinsic online nature of Data Stream settings makes competitive analysis an appropriate tool to approach such problems, especially when considering a decision-making under uncertainty and/or an economics perspective.

References

1. Aggarwal, C.: Data Streams: Models and Algorithms. Advances in Database Systems. Springer (2007)
2. Alon, N., Matias, Y., Szegedy, M.: The Space Complexity of Approximating the Frequency Moments. Journal of Computer and System Sciences 58(1), 137–147 (1999)
3. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Online auctions and generalized secretary problems. ACM SIGecom Exchanges 7(2), 1–11 (2008)
4. Becchetti, L., Chatzigiannakis, I., Giannakopoulos, Y.: Streaming techniques and data aggregation in networks of tiny artefacts. Computer Science Review 5(1), 27–46 (2011)
5. Becchetti, L., Koutsoupias, E.: Competitive Analysis of Aggregate Max in Windowed Streaming. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 156–170. Springer, Heidelberg (2009)
6. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
7. Boyer, R.S., Moore, J.S.: MJRTY-A Fast Majority Vote Algorithm. Technical report, Texas University at Austin, Insitute for Computing Science and Computer Applications (1981)
8. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. Proceedings of the VLDB Endowment 1(2), 1530–1541 (2008)
9. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM Journal on Computing, 635–644 (2002)
10. Ferguson, T.S.: Who solved the secretary problem? Statistical Science 4(3), 282–296 (1989)
11. Fiat, A., Woeginger, G.J. (eds.): Online Algorithms 1996. LNCS, vol. 1442. Springer, Heidelberg (1998)
12. Gama, J., Geber, M.M. (eds.): Learning from Data Streams: Processing Techniques in Sensor Networks. Springer (2007)
13. Liu, L., Tamer Ozsu, M. (eds.): Encyclopedia of Database Systems. Springer (2009)
14. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming 2(2), 143–152 (1982)
15. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press (2005)
16. Munro, J.I., Paterson, M.S.: Selection and sorting with limited storage. In: Proceedings of FOCS 1978, pp. 253–258 (1978)
17. Muthukrishnan, S.: Data streams: Algorithms and applications. Now Publishers Inc. (2005)
18. Yao, A.C.-C.: Probabilistic computations: Toward a unified measure of complexity. In: Proceedings of FOCS 1977, pp. 222–227 (1977)
19. Yi, K., Zhang, Q.: Multidimensional online tracking. ACM Trans. Algorithms 8(2), 12:1–12:16 (2012)

Kernel Bounds for Structural Parameterizations of Pathwidth*

Hans L. Bodlaender, Bart M.P. Jansen, and Stefan Kratsch

Utrecht University, The Netherlands
{h.l.bodlaender,b.m.p.jansen,s.kratsch}@uu.nl

Abstract. Assuming the AND-distillation conjecture, the Pathwidth problem of determining whether a given graph G has pathwidth at most k admits no polynomial kernelization with respect to k. The present work studies the existence of polynomial kernels for Pathwidth with respect to other, structural, parameters.

Our main result is that, unless NP $\subseteq$ coNP/poly, Pathwidth admits no polynomial kernelization even when parameterized by the vertex deletion distance to a clique, by giving a cross-composition from Cutwidth. The cross-composition works also for Treewidth, improving over previous lower bounds by the present authors. For Pathwidth, our result rules out polynomial kernels with respect to the distance to various classes of polynomial-time solvable inputs, like interval or cluster graphs.

This leads to the question whether there are nontrivial structural parameters for which Pathwidth does admit a polynomial kernelization. To answer this, we give a collection of graph reduction rules that are safe for Pathwidth. We analyze the success of these results and obtain polynomial kernelizations with respect to the following parameters: the size of a vertex cover of the graph, the vertex deletion distance to a graph where each connected component is a star, and the vertex deletion distance to a graph where each connected component has at most c vertices.

1 Introduction

The notion of kernelization provides a systematic way to mathematically analyze what can be achieved by (polynomial-time) preprocessing of combinatorial problems [1]. This paper discusses kernelization for the problem to determine the *pathwidth* of a graph. The notion of pathwidth was introduced by Robertson and Seymour in their fundamental work on graph minors [2], and is strongly related to the notion of treewidth. There are several notions that are equivalent to pathwidth including *interval thickness*, *vertex separation number*, and *node search number* (see [3] for an overview). The problem to determine the pathwidth of a graph is well studied, also under the different names of the problem.

* This work was supported by the Netherlands Organization for Scientific Research (N.W.O.), project "KERNELS: Combinatorial Analysis of Data Reduction".

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 352–363, 2012.

It is well known that the decision problem corresponding to pathwidth is NP-complete, even on restricted graph classes such as bipartite graphs and chordal graphs [4,5]. A commonly employed practical technique is therefore to preprocess the input before trying to compute the pathwidth, by employing a set of (reversible) data reduction rules. Similar preprocessing techniques for the TREEWIDTH problem have been studied in detail [6,7], and their practical use has been verified in experiments [8]. Using the concept of kernelization we may analyze the quality of such preprocessing procedures within the framework of parameterized complexity. A *parameterized problem* is a language $Q \subseteq \Sigma^* \times \mathbb{N}$, and such a problem is (strongly uniform) *fixed-parameter tractable* (FPT) if there is an algorithm that decides membership of an instance (x, k) in time $f(k)|x|^{\mathcal{O}(1)}$ for some computable function f. A *kernelization* (or *kernel*) for Q is a polynomial-time algorithm which transforms each input (x, k) into an *equivalent* instance (x', k') such that $|x'|, k' \leq g(k)$ for some computable function g, which is the *size* of the kernel. Kernels of polynomial size are of particular interest due to their practical applications. To analyze the quality of preprocessing rules for PATHWIDTH we therefore study whether they yield polynomial kernels for suitable parameterizations of the PATHWIDTH problem.

As the pathwidth of a graph equals the maximum of the pathwidth of its connected components, the PATHWIDTH problem with standard parameterization is AND-compositional and thus has no polynomial kernel unless the AND-distillation conjecture does not hold [9]. We thus do not expect to have kernels for PATHWIDTH of size polynomial in the target value for pathwidth k, and we consider whether polynomial kernels can be obtained with respect to other parameterizations.

As PATHWIDTH is known to be polynomial-time solvable when restricted graph classes such as interval graphs [3], trees [10] and cographs [11], it seems reasonable to think that determining the pathwidth of a graph G which is "almost" an interval graph should also be polynomial-time solvable. Formalizing the notion of "almost" as the number of vertices that have to be deleted to obtain a graph in the restricted class $\mathcal{F}$, we can study the extent to which data reduction is possible for graphs which are close to polynomial-time solvable instances through the following problem:

PATHWIDTH PARAMETERIZED BY A MODULATOR TO $\mathcal{F}$
Instance: A graph $G = (V, E)$, a positive integer k, and a set $S \subseteq V$ such that $G - S \in \mathcal{F}$.
Parameter: $\ell := |S|$.
Question: $\mathbf{pw}(G) \leq k$?

The set S is a *modulator* to the class $\mathcal{F}$. Observe that pathwidth should be polynomial-time solvable on $\mathcal{F}$ in order for this parameterized problem to be FPT. Our main result is a kernel lower bound for such a parameterization of PATHWIDTH. We prove that despite the fact that the pathwidth of an interval graph is simply the size of its largest clique minus one — which is very easy to find on interval graphs — the PATHWIDTH problem parameterized by a modulator to an interval graph does not admit a polynomial kernel unless

NP $\subseteq$ coNP/poly. In fact, we prove the stronger statement that, under the same condition, Pathwidth parameterized by a modulator to a single clique (i.e., by distance to $\mathcal{F}$ consisting of all complete graphs) does not admit a polynomial kernel[1] (Section 5). As the graph resulting from the lower-bound construction is co-bipartite, its pathwidth and treewidth coincide [12]: a corollary to our theorem therefore shows that Treewidth parameterized by vertex-deletion distance to a clique does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly, thereby significantly strengthening a result of our earlier work [6] where we only managed to prove kernel lower bounds by modulators from cluster graphs and co-cluster graphs.

Our kernel bound effectively shows that even in graphs which are cliques after the deletion of k vertices, the information contained in the (non)edges between these k vertices and the clique is such that we cannot decrease the size of the clique to polynomial in k in polynomial time, without changing the answer in some cases.

Faced with these negative results, we try to formulate *safe* reduction rules for Pathwidth (Section 3). It turns out that many of the rules for Treewidth (e.g., the rules involving (almost) simplicial vertices) are invalid when applied to Pathwidth, and more careful reduction procedures are needed to reduce the number of such vertices. We obtain several reduction rules for pathwidth, and show that they lead to provable data reduction guarantees when analyzed using a suitable parameterization (Section 4). In particular we prove that Pathwidth parameterized by a vertex cover S (i.e., using $\mathcal{F}$ as the class of edgeless graphs in the template above) admits a kernel with $\mathcal{O}(|S|^3)$ vertices, that the parameterization by a modulator S' to a disjoint union of stars has a kernel with $\mathcal{O}(|S'|^4)$ vertices, and finally that parameterizing by a set S'' whose deletion leaves a graph in which every connected component has at most c vertices admits a kernel with $\mathcal{O}(c \cdot |S''|^3 + c^2 \cdot |S''|^2)$ vertices.

2 Preliminaries

In this work all graphs are finite, simple, and undirected. The open neighborhood of a vertex $v \in V$ in a graph G is denoted by $N_G(v)$, and its closed neighborhood is $N_G[v]$. For sets of vertices $W \subseteq V$ we let $N_G[W] = \bigcup_{v \in W} N_G[v]$ and $N_G(W) = N_G[W] \setminus W$. If $S \subseteq V$ is a vertex set then $G - S$ denotes the graph obtained from G by deleting all vertices of S and their incident edges. For a single vertex v we write $G - v$ instead of $G - \{v\}$. A vertex v is *simplicial* in a graph G if $N_G(v)$ is a clique. A vertex $v \in V$ is *almost simplicial* in a graph G if v has a neighbor w such that $N_G(v) - \{w\}$ is a clique. In such a case, we call w the *special neighbor* of v. For a set of vertices $W \subseteq V$, the subgraph of G

[1] For completeness we point out that Pathwidth parameterized by a modulator to a clique is FPT: try all orderings in which the vertices from S can be introduced and forgotten in a decomposition, and do a polynomial-time computation for each ordering to find the best way to fit the clique $G - X$ into the decomposition.

induced by W is denoted as $G[W]$. A *path decomposition* of a graph $G = (V, E)$ is a non-empty sequence $(X_1, \dots, X_r)$ of subsets of V called *bags*, such that:

- $\bigcup_{1 \leq i \leq r} X_i = V$,
- for all edges $\{v, w\} \in E$ there is a bag X_i containing v and w, and
- for all vertices $v \in V$, the bags containing v are consecutive in the sequence.

The *width* of a path decomposition is $\max_{1 \leq i \leq r} |X_i| - 1$. The *pathwidth* $\mathbf{pw}(G)$ of G is the minimum width of a path decomposition of G. Throughout the paper we will often make use of the fact that the pathwidth of a graph does not increase when taking a minor. We also use the following results.

Lemma 1 (Cf. [11]). *If graph G contains a clique W then any path- or tree decomposition for G has a bag containing all vertices of W.*

Lemma 2. *All graphs G admit a minimum-width path decomposition in which each simplicial vertex is contained in exactly one bag of the decomposition.*

Proof. Lemma 1 shows that for each simplicial vertex v, any path decomposition of G has a bag containing the clique $N[v]$. As removal of v from all other bags preserves the validity of the decomposition, we may do so independently for all simplicial vertices to obtain a decomposition of the desired form. □

3 Reduction Rules

In this section we give a collection of reduction rules. Formally, each rule takes as input an instance (G, S, k) of Pathwidth parameterized by a modulator to $\mathcal{F}$, and outputs an instance (G', S', k'). With the exception of occasionally outright deciding **yes** or **no**, none of our reduction rules change the modulator S or the value of k. In the interest of readability we shall therefore be less formal in our exposition, and make no mention of the values of S' and k' in the remainder. We say that a rule is *safe for pathwidth* (or in short: safe) if for each input (G, S, k) and output (G', S', k'), the pathwidth of G is at most k if and only if the pathwidth of G' is at most k'. Any subset of the rules gives a 'safe' preprocessing algorithm for pathwidth: apply the rules until no longer possible. We will argue later that this takes polynomial time for our rules, and give kernel bounds for some parameters of the graphs.

3.1 Vertices of Small Degree

We start off with a few simple rules for vertices of small degree. Note that, necessarily, these rules are slightly more restrictive than for the treewidth case; e.g., we cannot simply delete vertices of degree one since trees have treewidth one but unbounded pathwidth. The first rule is trivial.

Rule 1. *Delete any vertex of degree zero.*

Rule 2. *If two degree-one vertices share their neighbor then delete one of them.*

Correctness of Rule 2 follows from insights on the pathwidth of trees, pioneered by Ellis et al. [10]. A self-contained proof is provided in the full version.

The following rule handles certain vertices of degree two; a correctness proof is given in the full version.

Rule 3. *Let v, w be two vertices of degree two, and suppose x and y are common neighbors to v and w with $x \in S$. Then remove w and add the edge $\{x, y\}$.*

3.2 Common Neighbors and Disjoint Paths

Rule 4 in this section also appears in our work on kernelization for treewidth [6] and traces back to well-known facts about treewidth (e.g.,[13,14]). It is also safe in the context of pathwidth; the safeness proof is identical to when dealing with treewidth and is hence deferred to the full version.

Lemma 3. *Let v and w be nonadjacent vertices. Suppose there are at least $k+1$ internally vertex disjoint paths from v to w in (V, E). Then the pathwidth of G is at most k, if and only if the pathwidth of $G' = (V, E \cup \{\{v, w\}\})$ is at most k.*

A special case of Lemma 3, and the implied Rule 4, is when v and w have at least $k+1$ common neighbors. As we do not want to increase the size of a modulator, we only add edges between pairs of vertices with at least one endpoint in the modulator; thus $G - S$ remains unchanged.

Rule 4 (Disjoint paths (with a modulator)). *Let $v \in S$ be nonadjacent to $w \in V$, and suppose there are at least $k+1$ paths from v to w that only intersect at v and w, where k denotes the target pathwidth. Then add the edge $\{v, w\}$.*

3.3 Simplicial Vertices

In this section, we give a safe rule that helps to bound the number of simplicial vertices of degree at least two in a graph. Recall that we already have rules for vertices of degree one and zero, which are trivially simplicial.

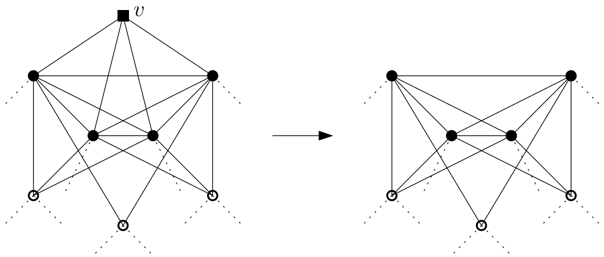


Fig. 1. An example of an application of the Simplicial vertex rule

Lemma 4. *Let $G = (V, E)$ be a graph, and let $v \in V$ be a simplicial vertex of degree at least two. If for all $x, y \in N_G(v)$ with $x \neq y$ there is a simplicial vertex $w \notin N_G[v]$ such that $x, y \in N_G(w)$, then $\mathbf{pw}(G) = \mathbf{pw}(G - v)$.*

Proof. As $G - v$ is a subgraph of G, we directly have that $\mathbf{pw}(G - v) \leq \mathbf{pw}(G)$. For the converse, let $(X_1, \ldots, X_r)$ be an optimal path decomposition of $G - v$. Using Lemma 2, we assume that for each simplicial vertex x, there is a unique bag X_{i_x} with $N_G[x] \subseteq X_{i_x}$.

Let $C = N_G(v)$. A bag that contains C is called a C-bag. As C is a clique, Lemma 1 shows there is at least one C-bag. The C-bags must be consecutive in the path decomposition; let them be $X_{i_1}, \ldots, X_{i_2}$. We will first show there is a vertex $w \notin N_G[v]$ which is simplicial in $G - v$, and is contained in a C-bag. Let $x, y \in C$ (possibly with $x = y$) be vertices such that x does not occur in bags with index smaller than i_1, and y does not occur in bags of index larger than i_2.

If $x \neq y$ then let $w \notin N_G[v]$ be simplicial in G such that $x, y \in N_G(w)$, whose existence is guaranteed by the preconditions. As w is also simplicial in $G - v$ it occurs in a unique bag, which must be a C-bag since it must meet its neighbors x and y there. If $x = y$ then, as v has degree at least two, there is a vertex $w \notin N_G[v]$ which is simplicial in G and adjacent to x; hence its unique occurrence is also in a C-bag.

Thus we have established there is a vertex $w \notin N_G[v]$ which is simplicial in $G - v$ and is contained in exactly one bag, which is a C-bag X_i. Now insert a new bag just after X_i, with vertex set $X_i - \{w\} \cup \{v\}$. As $X_i - \{w\}$ contains all v's neighbors, this gives a path decomposition of G without increasing the width, and concludes the proof. □

Lemma 4 directly shows that Rule 5 is safe for Pathwidth.

Rule 5. *For each $e \in E$, compute span(e) as the number of simplicial vertices that are adjacent to both endpoints of the edge. If $v \in V$ is a simplicial vertex of degree at least two such that each edge between a pair of neighbors of v has span at least 2, then remove v.*

3.4 Simplicial Components

Let S be the set of vertices used as the modulator. We say that a set of vertices W is a *simplicial component* if W is a connected component in $G-S$ and $N_G(W) \cap S$ is a clique. Our next rule deals with simplicial components.

Rule 6 (Simplicial components of known pathwidth). *Let $S \subseteq V$ be the modulator and let k denote the target pathwidth. Suppose that for each pair $v, w \in S \cap N_G(W)$ (including $v = w$), there are at least $2k+3$ simplicial components $Z \neq W$ such that $\{v, w\} \subseteq N_G(Z)$ and $\mathbf{pw}(G[Z]) \geq \mathbf{pw}(G[W])$. Then remove W and its incident edges.*

Note that we have to include the case $v = w$ to ensure correctness for simplicial components which are adjacent to exactly one vertex in the modulator. The safeness proof for Rule 6 is given in the full version.

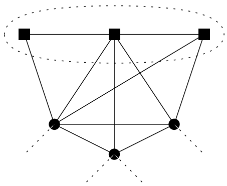


Fig. 2. The vertices marked with a square box form a simplicial component

Let us briefly discuss the running time of this reduction rule. As the modulator ensures that $G - S$ is contained in the graph class $\mathcal{F}$, the rule can be applied in polynomial time if the pathwidth of graphs in $\mathcal{F}$ can be determined efficiently. In the setting in which we apply the rule, the graphs in $\mathcal{F}$ are either disjoint unions of stars (which are restricted types of forests, allowing the use of the linear-time algorithm of Ellis et al. [10]), or $\mathcal{F}$ has constant pathwidth which means that the FPT algorithm for k-Pathwidth [13] runs in linear time.

3.5 Almost Simplicial Vertices

For almost simplicial vertices, we have a rule that replaces an almost simplicial vertex by a number of vertices of degree two. In several practical settings, the increase of number of vertices may be undesirable; the rule is useful to derive some theoretical bounds.

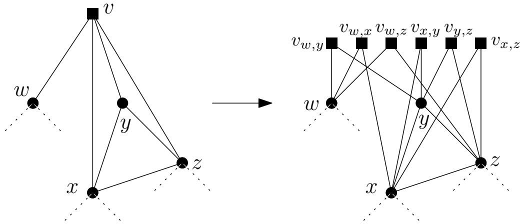


Fig. 3. An example of an application of the rule for almost simplicial vertices

Lemma 5. *Let $G = (V, E)$ be a graph and let $v \in V$ be an almost simplicial vertex of degree at least three, with special neighbor w. Let G' be obtained by deleting v and by adding a vertex $v_{p,q}$ with neighbors p and q for any $p, q \in N_G(v)$ with $p \neq q$. Then $\mathbf{pw}(G) = \mathbf{pw}(G')$.*

The proof of the lemma is postponed to the full version. The lemma justifies the following reduction rule, by observing that an almost simplicial vertex v with $\deg_G(v) > k + 1$ means that $\mathbf{pw}(G) > k$, as $N_G[v] - w$ then forms a clique of size at least $k + 2$.

Rule 7. *Let $v \in V \setminus S$ be an almost simplicial vertex of degree at least three with special neighbor w. Let k be the target pathwidth. If $\deg_G(v) > k+1$ then output **no**. Otherwise, delete v and add a vertex $v_{p,q}$ with neighbors p and q for any $p, q \in N(v)$ with $p \neq q$.*

As a simplicial vertex is trivially almost simplicial, note that — in comparison to Rule 5 — the previous rule gives an alternative way of dealing with simplicial vertices.

4 Polynomial Kernelizations

For each of the safe rules given in the previous section, there is a polynomial time algorithm that tests if the rule can be applied, and if so, modifies the graph accordingly. (We assume that for Rule 6 the bound ℓ on the pathwidth of the components is a constant.) The following lemma shows that any algorithm that exhaustively applies (possibly just a subset of) these reduction rules can be implemented to run in polynomial time.

Lemma 6. *Each input instance (G, S, k) is exhaustively reduced by $\mathcal{O}(n^2 + nk^2)$ applications of the reduction rules.*

Proof. First we note that for non-trivial instances, Rule 4 does not add edges to a vertex of degree at most two. In particular, no rule increases the number of vertices of degree at least three. So, we have at most n applications of a rule that removes a vertex of degree at least three, and $\mathcal{O}(n^2)$ applications of Rule 4. Rule 7 is therefore executed at most n times in total, and thus the number of vertices of degree two that are added in these steps is bounded by $\mathcal{O}(nk^2)$. As each other rule removes at least one vertex, the total number of rule applications in G is bounded by $\mathcal{O}(n^2 + nk^2)$. □

By analyzing our reduction rules with respect to different structural parameters, we get the following results.

Theorem 1. Pathwidth parameterized by a modulator to $\mathcal{F}$ *admits polynomial kernels for the following choices of $\mathcal{F}$:*

1. *A kernel with $\mathcal{O}(\ell^3)$ vertices when $\mathcal{F}$ is the class of all independent sets, i.e., if the modulator S is a vertex cover.*
2. *A kernel with $\mathcal{O}(c \cdot \ell^3 + c^2 \cdot \ell^2)$ vertices when $\mathcal{F}$ is the class of all graphs with connected components of size at most c.*
3. *A kernel with $\mathcal{O}(\ell^4)$ vertices when $\mathcal{F}$ is the class of all disjoint unions of stars.*

Proof. We show Part 3 followed by Part 2. Part 1 follows from the latter since it is a special case corresponding to $c = 1$.

(Part 3.) As stars have pathwidth one, graphs with a modulator S of size ℓ to a set of stars have pathwidth at most $\ell + 1$. Thus, if $k \geq \ell + 1$, we return a dummy **yes**-instance of constant size. Now, assume $k \leq \ell$.

Our kernelization applies Rules 1–6 while possible, and applies Rule 7 to all vertices which have at most one neighbor in $G-S$. (Applying the rule to vertices with more neighbors in $G-S$ might cause the resulting graph $G'-S$ not to be a disjoint union of stars.) Recall for Rule 6 that $\mathbf{pw}(G-S) \leq 1$.

Let (G, S, k) be a reduced instance. We will first bound the number of connected components of $G-S$, with separate arguments for simplicial and nonsimplicial components. Each component is a star, i.e., it is a single vertex or a $K_{1,r}$ for some r (a center vertex with r leaves). Note that in this proof the term leaf refers to a leaf of a star in $G-S$, independent of its degree in G (and all degrees mentioned are with respect to G).

Associate each nonsimplicial component C of $G-S$ to an arbitrary pair of nonadjacent neighbors of C in S. It is easy to see that each such component provides a path between the two chosen neighbors, and that for different components these paths are internally vertex disjoint. Thus, since Rule 4 does not apply, no pair of vertices of S has more than k components associated to it. Hence there are at most $k \cdot |S|^2 = \mathcal{O}(\ell^3)$ nonsimplicial components.

Now consider a simplicial component W of $G-S$, and note that $\mathbf{pw}(G[W]) \leq 1$. As Rule 6 does not apply, there is a pair $v, w \in S \cap N_G(W)$ (possibly $v = w$) such that there are strictly less than $2k+3$ simplicial components $W' \neq W$ with $\mathbf{pw}(G[W']) \geq \mathbf{pw}(G[W])$ and $\{v, w\} \subseteq N_G(W')$. Associate W to the pair v, w. It follows immediately that no pair of vertices of S has more than $2k+3$ components associated to it, which gives a bound of $(2k+3) \cdot |S|^2 = \mathcal{O}(\ell^3)$ on the number of simplicial components.

Thus we find that $G-S$ has a total of $\mathcal{O}(\ell^3)$ connected components (each of which is a star). This bounds the number of centers of stars by $\mathcal{O}(\ell^3)$. It remains to bound the total number of leaves that are adjacent to those centers.

Clearly, each star center has at most one leaf which has degree one (in G). Each leaf of degree two has exactly one neighbor in S in addition to its adjacent star center. Since Rule 3 does not apply, no two leaves of degree two can have the same star center and neighbor in S; thus there are at most $\mathcal{O}(\ell^4)$ leaves of degree two.

Now, we are going to count the number of leaves (of stars) that are of degree more than two. For each such leaf, one neighbor is the center of its star and all other neighbors are in S. If its neighbors in S would form a clique, then the leaf would be almost simplicial in G (with the star center as the special neighbor) and Rule 7 would apply. Hence, as G is reduced, we can associate each such leaf to a nonadjacent pair of vertices in S. As Rule 4 cannot be applied, we associate $\mathcal{O}(k)$ vertices to a pair, and thus the number of such leaves is bounded by $\mathcal{O}(k \cdot \ell^2) = \mathcal{O}(\ell^3)$.

Thus, the total number of vertices in G is bounded by $\mathcal{O}(\ell^4)$. By Lemma 6 the reduction rules can exhaustively be applied in polynomial time. As the rules preserve the fact that $G-S$ is a disjoint union of stars, the resulting instance is a correct output for a kernelization algorithm. This completes the proof of Part 3.

(Part 2.) Fix some constant c and let $\mathcal{F}$ be the class of all graphs of component size at most c. Let (G, S, k) be an input instance. Note that the pathwidth of G is bounded by $c + |S| - 1$, since each component of $G - S$ has pathwidth at most $c - 1$. We assume that $k \leq c + |S| - 2$; otherwise the instance is **yes** and we may return a dummy **yes**-instance of constant size.

Our algorithm uses Rules 1, 2, 4, 5, and 6. Consider a graph G where none of these rules can be applied. The bounds for the number of simplicial and nonsimplicial components of $G-S$ work analogously to Part 3; there are $\mathcal{O}(k|S|^2)$ components of the respective types. This gives a total of $\mathcal{O}(|S|+c\cdot(c+|S|)\cdot|S|^2) = \mathcal{O}(c^2|S|^2 + c|S|^3)$ vertices in G, using that $k \leq c + |S| - 2$. This completes the proof of Part 2. □

5 Lower Bounds: Modulator to a Single Clique

We show that the problems Treewidth parameterized by a modulator to a single clique (TWMSC) and Pathwidth parameterized by a modulator to a single clique (PWMSC) do not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. In fact, we show that the results hold when restricted to cobipartite graphs; as for these graphs the pathwidth equals the treewidth [12], the same proof works for both problems. The problems are covered by the general template given in the introduction, when using $\mathcal{F}$ as the class of all cliques.

To prove the lower bound we employ the technique of cross-composition [15], starting from the following NP-complete version [16, Corollary 2.10] of the Cutwidth problem:

Cutwidth on cubic graphs (CUTWIDTH3)
Instance: A graph G on n vertices in which each vertex has degree at least one and at most three, and an integer $k \leq |E(G)|$.
Question: Is there a linear layout of G of cutwidth at most k, i.e., a permutation π of $V(G)$ such that $\max_{i=1}^{n} |\{\{u, v\} \in E(G) \mid \pi(u) \leq i < \pi(v)\}| \leq k$?

As space restrictions prohibit us from presenting the full proof in this extended abstract, we will sketch the main ideas. To obtain a kernel lower bound through cross-composition, we have to embed the logical OR of a series of t input instances of CUTWIDTH3 on n vertices each into a single instance of the target problem for a parameter value polynomial in $n + \log t$. At the heart of our construction lies an idea of Arnborg et al. [4] employed in their NP-completeness proof for Treewidth. They interpreted the treewidth of a graph as the minimum cost of an elimination ordering on its vertices[2], and showed how for a given graph G a co-bipartite graph G^* can be created such that the cost of elimination orderings on G^* corresponds to the cutwidth of G under a related ordering.

[2] To eliminate a vertex in a graph means to remove it while completing its open neighborhood into a clique. When eliminating the vertices of a graph in the order given by π, the cost of the elimination ordering π is the maximum degree of a vertex at the time it is eliminated.

We extend their construction significantly. By the degree bound, instances with n vertices have $\mathcal{O}(n^2)$ different degree sequences. The framework of cross-composition thus allows us to work on instances with the same degree sequence (and same k). By enforcing that the structure of one side of the co-bipartite graph G^* only has to depend on this sequence, all inputs can share the same "right hand side" of the co-bipartite graph; this part will remain small and act as the modulator. By a careful balancing act of weight values we then ensure that the cost of elimination orderings on the constructed graph G^* are dominated by eliminating the vertices corresponding to exactly one of the input instances, ensuring that a sufficiently low treewidth is already achieved when one of the input instances is **yes**. On the other hand, the use of a binary-encoding representation of instance numbers ensures that low-cost elimination orderings for G^* do not mix vertices corresponding to different input instances. The remaining details can be found in the full version of this paper. Our construction yields the following results.

Theorem 2. *Unless NP $\subseteq$ coNP/poly,* Pathwidth *and* Treewidth *do not admit polynomial kernels when parameterized by a modulator to a single clique.*

Interestingly, the parameter at hand is nothing else than the size of a vertex cover in the complement graph.

6 Conclusions

In this paper, we investigated the existence of polynomial kernelizations for Pathwidth. Taking into account that the problem is already known to be AND-compositional with respect to the target pathwidth — thus excluding polynomial kernels under the AND-distillation conjecture — we study alternative, structural parameterizations.

Our main result is that Pathwidth admits no polynomial kernelization with respect to the number of vertex deletions necessary to obtain a clique, unless NP $\subseteq$ coNP/poly. This rules out polynomial kernels for vertex deletion distance from various interesting graph classes on which Pathwidth is known to be polynomial-time solvable, like chordal and interval graphs.

On the positive side we develop a collection of safe reduction rules for Pathwidth. Analyzing the effect of the rules we show that they give polynomial kernels with respect to the following parameters: vertex cover (i.e., distance from the class of independent sets), distance from graphs of bounded component size, and distance from disjoint union of stars.

It is an interesting open problem to determine whether there is a polynomial kernel for Pathwidth parameterized by the size of a feedback vertex set. For the related Treewidth problem, a kernel with $\mathcal{O}(|S|^4)$ vertices is known [6], where S denotes a feedback vertex set. Regarding Pathwidth, long paths in $G - S$ are the main obstacle that needs to be addressed by additional reduction rules.

References

1. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
2. Robertson, N., Seymour, P.D.: Graph minors. I. Excluding a forest. J. Comb. Theory, Ser. B 35, 39–61 (1983)
3. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. 209(1-2), 1–45 (1998)
4. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic and Discrete Methods 8, 277–284 (1987)
5. Gustedt, J.: On the pathwidth of chordal graphs. Discrete Appl. Math. 45, 233–248 (1993)
6. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 437–448. Springer, Heidelberg (2011)
7. van den Eijkhof, F., Bodlaender, H.L., Koster, A.M.C.A.: Safe reduction rules for weighted treewidth. Algorithmica 47(2), 139–158 (2007)
8. Bodlaender, H.L., Koster, A.M.C.A., van den Eijkhof, F.: Preprocessing rules for triangulation of probabilistic networks. Computational Intelligence 21(3), 286–305 (2005)
9. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
10. Ellis, J.A., Sudborough, I.H., Turner, J.S.: The vertex separation and search number of a graph. Inf. Comput. 113(1), 50–79 (1994)
11. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. SIAM J. Discrete Math. 6, 181–188 (1993)
12. Möhring, R.H.: Triangulating graphs without asteroidal triples. Discrete Applied Mathematics 64(3), 281–287 (1996)
13. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
14. Clautiaux, F., Carlier, J., Moukrim, A., Négre, S.: New Lower and Upper Bounds for Graph Treewidth. In: Jansen, K., Margraf, M., Mastrolli, M., Rolim, J.D.P. (eds.) WEA 2003. LNCS, vol. 2647, pp. 70–80. Springer, Heidelberg (2003)
15. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Cross-composition: A new technique for kernelization lower bounds. In: Proc. 28th STACS, pp. 165–176 (2011)
16. Monien, B., Sudborough, I.H.: Min cut is NP-complete for edge weighted trees. Theor. Comput. Sci. 58, 209–229 (1988)

Kernel Lower Bounds Using Co-nondeterminism: Finding Induced Hereditary Subgraphs

Stefan Kratsch[1,⋆], Marcin Pilipczuk[2,⋆⋆],
Ashutosh Rai[3], and Venkatesh Raman[3]

[1] Utrecht University, Utrecht, The Netherlands
s.kratsch@uu.nl
[2] Institute of Informatics, University of Warsaw, Poland
malcin@mimuw.edu.pl
[3] The Institute of Mathematical Sciences, Chennai, India
{ashutosh,vraman}@imsc.res.in

Abstract. This work further explores the applications of co-nondeterminism for showing kernelization lower bounds. The only known example excludes polynomial kernelizations for the Ramsey problem of finding an independent set or a clique of at least k vertices in a given graph (Kratsch 2012, SODA). We study the more general problem of finding induced subgraphs on k vertices fulfilling some hereditary property Π, called Π-Induced Subgraph. The problem is NP-hard for all non-trivial choices of Π by a classic result of Lewis and Yannakakis (JCSS 1980). The parameterized complexity of this problem was classified by Khot and Raman (TCS 2002) depending on the choice of Π. The interesting cases for kernelization are for Π containing all independent sets and all cliques, since the problem is trivial or W[1]-hard otherwise.

Our results are twofold. Regarding Π-Induced Subgraph, we show that for a large choice of natural graph properties Π, including chordal, perfect, cluster, and cograph, there is no polynomial kernel with respect to k. This is established by two theorems: one using a co-nondeterministic variant of cross-composition and one by a polynomial parameter transformation from Ramsey.

Additionally, we show how to use improvement versions of NP-hard problems as source problems for lower bounds, without requiring their NP-hardness. E.g., for Π-Induced Subgraph our compositions may assume existing solutions of size $k-1$. We believe this to be useful for further lower bound proofs, since improvement versions simplify the construction of a disjunction (OR) of instances required in compositions. This adds a second way of using co-nondeterminism for lower bounds.

⋆ Supported by the Netherlands Organization for Scientific Research (N.W.O.), project "KERNELS: Combinatorial Analysis of Data Reduction".

⋆⋆ Partially supported by NCN grant N206567140 and Foundation for Polish Science.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 364–375, 2012.

1 Introduction

The study of polynomial kernelization and, in particular, techniques for ruling out polynomial kernelizations has turned into one of the most well-studied directions in parameterized complexity [1,2,3,4,5,6,7,8] (see Section 2 for basic definitions). Almost all kernelization lower bound results and tools are, directly or indirectly, based on a framework due to Bodlaender et al. [1] and Fortnow and Santhanam [6]. On a high level the framework centers around the fact that NP-hard problems cannot have both a polynomial kernelization and a so-called composition algorithm unless NP $\subseteq$ coNP/poly (known to cause a collapse of the polynomial hierarchy). Chen and Mueller were the first to observe that the consequence still holds when both the kernelization and the composition are allowed to use co-nondeterminism, which is implicit in the work of Fortnow and Santhanam (cf. [9]). The aim of this paper is to explore the use of co-nondeterminism for the purpose of showing lower bounds for kernelization.

The only lower bound result so far based on a co-nondeterministic composition was given recently by Kratsch [8]. It is showed that the problem of finding an independent set *or* a clique of size at least k in a given graph does not admit a polynomial kernel with respect to k; we call the problem Ramsey for its relation to Ramsey's theorem. The composition used in the lower bound proof [8] relies on embedding instances of an improvement version of Ramsey into a so-called host graph; co-nondeterminism is used to, essentially, find such a graph.

Ramsey is a special case of the Π-Induced Subgraph problem (defined below) whose parameterized complexity was determined by Khot and Raman [10]. This work seeks to develop kernelization lower bounds for Π-Induced Subgraph (when parameterized by the solution size k), hoping to find further applications of co-nondeterminism for kernelization lower bounds.

Finding Induced Π-subgraphs. For a hereditary (i.e., closed under taking induced subgraphs) graph property Π, the Π-Induced Subgraph problem asks for the largest induced subgraph in the given graph G that belongs to the class Π. A classical result by Lewis and Yannakakis [11] from 1980 asserts that Π-Induced Subgraph is NP-hard for any non-trivial hereditary property Π.

Π-Induced Subgraph **Parameter:** k.
Input: A graph G and an integer k.
Question: Does there exist an induced subgraph of G on k vertices that belongs to Π?

Note that if Π contains all independent sets and all cliques, Π-Induced Subgraph is fixed-parameter tractable and admits a kernel of exponential size: by Ramsey's theorem, if G is too large, it contains a clique or an independent set of size k. If Π excludes both cliques and independent sets of a certain size then, by Ramsey's theorem, Π is finite and the problem is trivial. Khot and Raman [10] proved that for all other graph classes Π, Π-Induced Subgraph becomes W[1]-hard. Given this characterization we study the existence of polynomial kernelizations when Π is restricted to contain all independent sets and cliques.

Our Work. Regarding Π-Induced Subgraph we show that for most natural graph classes Π, including cographs, chordal, interval, split, perfect, and cluster graphs, there is no polynomial kernelization unless NP $\subseteq$ coNP/poly. This is proved by lower bounds for two classes of choices for Π which are established by a co-nondeterministic cross-composition and a parameterized reduction from Ramsey respectively.

As one tool for our compositions we establish a nice trick for allowing easier source problems which should be of independent interest for other lower bounds. We show that for two general classes of problems, modeled after monotone and anti-monotone optimization problems, it suffices to start from *improvement versions*, where each instance comes with a guaranteed solution which is only off by one from the target value. For example, we define Improvement Π-Induced Subgraph as the Π-Induced Subgraph problem, with an additional set $X \subseteq V(G)$ given in the input, that satisfies $|X| = k - 1$ and $G[X] \in \Pi$.

Since the breakthrough paper of Bodlaender et al. [1] it has been known that improvement versions are useful source problems for deriving compositions. Unfortunately this comes at a price: The framework requires to show NP-hardness of the improvement version; this may be straightforward, but it can be "a tough nut" or outright impossible. By introducing co-nondeterminism also into this part of the picture we are able to show NP-hardness *under co-nondeterministic many-one reductions* (see Section 3 for a precise definition) and establish that this is sufficient for all existing variants of the framework.

The Erdős-Hajnal Conjecture. Surprisingly, the question of kernelizations for Π-Induced Subgraph turns out to be related to the well-known Erdős-Hajnal conjecture about Ramsey bounds in hereditary graph classes. For an undirected graph G, let hom(G) denote the largest size of any clique or independent set of G. The well-known Ramsey's theorem asserts that $\hom(G) \geq \frac{1}{2}\log|V(G)|$ [12], and with high probability $\hom(G) = \mathcal{O}(\log|V(G)|)$ for random graphs $G(n, \frac{1}{2})$ [13]. Clearly the randomized argument for the upper bound fails if we assume that G belongs to some fixed non-trivial hereditary property Π. Erdős and Hajnal conjectured in 1989 that indeed forbidding a fixed induced subgraph may change the behavior of hom(G) dramatically.

Conjecture 1 ([14]). For every graph H there exists a constant $\varepsilon(H) > 0$ such that if G does not contain H as an induced subgraph, then $\hom(G) \geq |V(G)|^{\varepsilon(H)}$.

On the one hand, a proof of this conjecture would give polynomial bounds for Ramsey numbers when restricted to any hereditary class Π of graphs. On the other hand, it implies exponential lower bounds for the minimum number of vertices required to force the existence of a Π subgraph of a certain size.

Related Work. An alternative way of parameterizing Π-Induced Subgraph is to use the complement parameter, i.e., the number of vertices to be deleted to result in a graph in Π. Cai [15] has shown that this problem is fixed-parameter tractable and admits a polynomial sized kernel when Π forbids a finite set of induced subgraphs. A complete characterization is open for hereditary Π with an infinite forbidden set, though parameterized results are known for specific properties, e.g., [16,17,18,19].

2 Preliminaries

Graphs and the Erdős-Hajnal Property. We use standard graph notation, see e.g. [20]. We refer to [21] for the definitions of some of the graph classes. A family Π of graphs is said to be *non-trivial* if Π and its complement are infinite.

Definition 1. *A hereditary graph class Π has the* Erdős-Hajnal property *if there exists $\varepsilon(\Pi) > 0$ such that every $G \in \Pi$ has* $\hom(G) \geq |V(G)|^{\varepsilon(\Pi)}$*. I.e., every graph in Π has a clique or an independent set of size at least $|V(G)|^{\varepsilon(\Pi)}$.*

Lemma 1 (folklore, ♠[1]). *The class of perfect graphs, as well as any of its hereditary subclasses, has the Erdős-Hajnal property.*

Although Conjecture 1 is open, its statement is proven for many graphs H. One of the most important partial results is one due to Alon et al. [22].

Definition 2. *Let G be a graph and $(H_x)_{x \in V(G)}$ be a family of graphs, one for each vertex of G. We define the graph $Embed(G; (H_x)_{x \in V(G)})$ as the graph obtained from G by replacing each vertex x with the graph H_x. Formally,*

$$\begin{aligned} V(Embed(G; (H_x)_{x \in V(G)})) &= \{v(x,u) : x \in V(G), u \in V(H_x)\}, \\ E(Embed(G; (H_x)_{x \in V(G)})) &= \{v(x,u)v(x,w) : x \in V(G), uw \in E(H_x)\} \\ &\quad \cup \{v(x,u)v(y,w) : xy \in E(G), u \in V(H_x), w \in V(H_y)\} \end{aligned}$$

We say that $Embed(G; (H_x)_{x \in V(G)})$ is obtained by embedding $(H_x)_{x \in V(G)}$ into *G. We say that a hereditary class Π is* closed under embedding *if for all $G \in \Pi$ and $H_x \in \Pi$, for $x \in V(G)$, the graph $Embed(G; (H_x)_{x \in V(G)})$ belongs to Π.*

Theorem 1 ([22]). *Let G be a graph and $(H_x)_{x \in V(G)}$ be a family of graphs. If the classes of G-free graphs and H_x-free graphs, for all $x \in V(G)$, satisfy the Erdős-Hajnal property, then the class of $Embed(G; (H_x)_{x \in V(G)})$-free graphs also satisfies the Erdős-Hajnal property.*

Corollary 1 (♠). *Let Π be a hereditary property that forbids $K_{\ell,\ell}$ for some integer ℓ. Then Π satisfies the Erdős-Hajnal property.*

Proposition 1 ([23], ♠). *Cographs, perfect graphs, permutation graphs, weakly chordal graphs and AT-free graphs are closed under embedding.*

Parameterized Complexity and Kernelization. In the parameterized complexity setting, an instance comes with an integer parameter k — formally, a parameterized problem Q is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. We say that the problem is *fixed parameter tractable* (*FPT*) if there exists an

[1] Proofs of results marked with ♠ are deferred to the full version of the paper.

algorithm solving any instance (x, k) in time $f(k)\text{poly}(|x|)$ for some (usually exponential) computable function f. It is known that a problem is FPT if and only if it is kernelizable: a kernelization algorithm for a problem Q takes an instance (x, k) and in time polynomial in $|x| + k$ produces an equivalent instance (x', k') (i.e., $(x, k) \in Q$ iff $(x', k') \in Q$) such that $|x'| + k' \leq g(k)$ for some computable function g. The function g is the *size of the kernel*, and if it is polynomial, we say that Q admits a polynomial kernel.

Kernelization Hardness Framework. The two main ways of showing kernelization lower bounds are to either give some variant of composition or to transfer a lower bound from another problem to the target problem by an appropriate reduction. Next we provide the essential definitions for both strategies.

Definition 3 (polynomial equivalence relation [2]). *An equivalence relation $\mathcal{R}$ on Σ^* is called a* polynomial equivalence relation *if the following holds:*

1. *Equivalence of any $x, y \in \Sigma^*$ can be checked in time polynomial in $|x| + |y|$.*
2. *Any finite set $S \subseteq \Sigma^*$ has at most $(\max_{x \in S} |x|)^{\mathcal{O}(1)}$ $\mathcal{R}$-equivalence classes.*

The idea behind polynomial equivalence relations is that it suffices to give cross-compositions that work for any single equivalence class.

Definition 4 (co-nondeterministic cross-composition). *Let $L \subseteq \Sigma^*$ and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L* coNP-cross-composes *into Q if there is a polynomial equivalence relation $\mathcal{R}$ and a co-nondeterministic algorithm which, given t strings $x_1, x_2, \ldots, x_t$ belonging to the same equivalence class of $\mathcal{R}$, computes on each computation path an instance $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that:*

1. *if $x_i \in L$ for some $i \in \{1, \ldots, t\}$ then each computation path returns an instance $(x^*, k^*) \in Q$,*
2. *if $x_i \notin L$ for all $i \in \{1, \ldots, t\}$ then at least one computation path returns an instance $(x^*, k^*) \notin Q$,*
3. *and k^* is bounded by a polynomial in $(\max_{1 \leq i \leq t} |x_i| + \log t)$ in each output (x^*, k^*).*

The following theorem is an easy consequence of combining results for cross-composition [2] and coNP-composition [8]. The key fact is that a co-nondeterministic cross-composition of some language L into a parameterized problem Q combined with a polynomial kernelization for Q would give a *co-nondeterministic* weak distillation for L. As observed by Chen and Mueller the proof technique of Fortnow and Santhanam [6] still applies for co-nondeterministic distillations and implies $L \in \text{coNP/poly}$.

Theorem 2. *If $L \subseteq \Sigma^*$ has a co-nondeterministic cross-composition into the parameterized problem Q and Q has a polynomial kernel then $L \in$ coNP/poly. If L is NP-hard then NP $\subseteq$ coNP/poly.*

The second lower bound strategy, instead of a direct proof via compositions, is to provide a polynomial parameter transformation from a problem with an established lower bound.

Definition 5 ([24]). *Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems. We say that a polynomially computable function $f\colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ is a* polynomial parameter transformation *(PPT) from P to Q if for all $(x, k) \in \Sigma^* \times \mathbb{N}$ the following holds: $(x, k) \in P$ if and only if $(x', k') = f(x, k) \in Q$ and $k' \le k^{\mathcal{O}(1)}$.*

Theorem 3 ([24,5]). *Let P and Q be parameterized problems and $\tilde{P}$ and $\tilde{Q}$ be the unparameterized versions of P and Q respectively. Suppose that $\tilde{P}$ is NP-hard and $\tilde{Q}$ is in NP. Assume that there is a polynomial parameter transformation from P to Q. Then if Q admits a polynomial kernel, so does P. (Hence if P admits no polynomial kernel under some assumption then neither does Q.)*

3 Co-nondeterminism and Improvement Versions

In this section we show how to use improvement versions of NP-hard problems in compositions, without requiring them to be NP-hard themselves, but requiring a weaker assumption of NP-hardness under *co-nondeterministic many-one reductions* instead. We include here a proof for special case of Improvement Π-Induced Subgraph; a general result is deferred to the full version.

Definition 6. *Let $L, L' \subseteq \Sigma^*$ be two languages. We say that a nondeterministic polynomially computable function f is a* co-nondeterministic many-one reduction *from L to L' if the following holds:*

1. *if $x \in L$, then on all computation paths $f(x) \in L'$;*
2. *if $x \notin L$, then there exists a computation path with $f(x) \notin L'$.*

Proposition 2 (♠). *Let L and L' be languages with $L \in$ coNP/poly. If there is a co-nondeterministic many-one reduction from L' to L then $L' \in$ coNP/poly.*

Proposition 3. *Let Π be a hereditary graph class for which membership can be tested in deterministic polynomial time. Then there exists a (polynomial-time) co-nondeterministic many-one reduction from Π-Induced Subgraph to Improvement Π-Induced Subgraph.*

Proof. Let (G, k) be an instance of Π-Induced Subgraph. The reduction will guess an integer $k' \in \{1, \ldots, k\}$ and a set X of $k' - 1$ vertices. Then it checks in deterministic polynomial time whether the subgraph induced by X is contained in Π, i.e., if $G[X] \in \Pi$. If this is not the case then a dummy YES-instance of Improvement Π-Induced Subgraph is returned. Otherwise, the instance (G, X, k') is returned. This completes the description of the reduction.

Now, if (G, k) is a YES-instance, then it is easy to see that each path returns a YES-instance of Improvement Π-Induced Subgraph: The guessed set X is returned only if it induces a graph in Π, and G contains a Π-subgraph on k' vertices for all $k' \in \{1, \ldots, k\}$.

If (G, k) is a NO-instance then consider the minimum value $k' \in \{1, \ldots, k\}$ such that (G, k') is a NO-instance too; this value is guessed in one branch of the computation. Clearly G contains a Π-subgraph on $k' - 1$ vertices. Hence, one of the computation paths successfully guesses a subset X of $k' - 1$ vertices such that $G[X] \in \Pi$. On this path the instance (G, X, k') is returned which can be easily seen to be a NO-instance of Improvement Π-Induced Subgraph. □

Theorem 4. *Let Π be a non-trivial hereditary graph class for which membership can be tested in deterministic polynomial time. If there is a coNP-cross-composition from* Improvement Π-Induced Subgraph *to* Π-Induced Subgraph *and* Π-Induced Subgraph *has a polynomial kernel then* Π-Induced Subgraph $\in$ *coNP/poly and NP $\subseteq$ coNP/poly.*

Proof. It follows from Theorem 2 that the existence of both a coNP-cross-composition from Improvement Π-Induced Subgraph to Π-Induced Subgraph and a polynomial kernelization for Π-Induced Subgraph implies that Improvement Π-Induced Subgraph $\in$ coNP/poly. By Propositions 2 and 3, we have Π-Induced Subgraph $\in$ coNP/poly and, by NP-hardness of Π-Induced Subgraph, that NP $\subseteq$ coNP/poly. □

4 Kernelization Hardness for Some Π-Induced Subgraph Problems

The following theorem is the main result of this section.

Theorem 5. *Unless NP $\subseteq$ coNP/poly, there does not exist a kernelization algorithm with polynomial guarantee on the output size for* Π-Induced Subgraph *for any non-trivial hereditary graph class Π that is polynomially recognizable, contains all independent sets and cliques, is closed under embedding and has the Erdős-Hajnal property.*

This theorem, for example, covers perfect graphs and permutation graphs, as these classes are closed under embedding due to Proposition 1; recall that all subclasses of the perfect graphs have the Erdős-Hajnal property. However, split graphs and chordal graphs are not covered by this theorem as they are not closed under embedding. See the conclusions for a partial catalogue of graph classes covered by our results.

By Theorem 4, it suffices to show the following lemma to prove Theorem 5.

Lemma 2. *If Π is a hereditary graph class that is polynomially recognizable, contains all cliques and independent sets, is closed under embedding, and satisfies the Erdős-Hajnal property, then there exists a coNP-cross-composition algorithm from* Improvement Π-Induced Subgraph *into* Π-Induced Subgraph.

We now proceed to the description of the coNP-cross-composition algorithm for Improvement Π-Induced Subgraph towards proving Lemma 2. We start by preparing a host graph, similar to the case of the Ramsey problem [8]. Fix a graph class Π satisfying the assumptions of Lemma 2 and let $\varepsilon > 0$ such that any $G \in \Pi$ satisfies $\hom(G) \geq |V(G)|^{\varepsilon}$.

Recall that the classical Ramsey number $R(\ell)$ is defined as the smallest integer such that any graph on $R(\ell)$ vertices contains an independent set or a clique of size ℓ. We define Π-Ramsey numbers as follows: for a positive integer ℓ, let $R_{\Pi}(\ell)$ be the smallest integer such that any graph on at least $R_{\Pi}(\ell)$ vertices contains an induced subgraph from Π of size ℓ. Note that this is well-defined, as all cliques and independent sets belong to Π, and $R_\Pi(\ell) \leq R(\ell)$. We continue with a simple lemma about gaps in Π-Ramsey numbers, similar to the corresponding lemma about Ramsey numbers in [8].

Lemma 3 (♠). *There exists a constant $\delta(\Pi)$ that depends on the class Π only, such that for any integer $t > \delta(\Pi)$, there exists a positive integer $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$, such that $R_\Pi(\ell+1) > R_\Pi(\ell) + t$.*

Now we describe the co-nondeterministic construction of a host graph, which will later be extended to a coNP-cross-composition.

Lemma 4. *There exists a nondeterministic algorithm that, given an integer $t > \delta(\Pi)$, in time polynomial in t either answers* `FAIL` *or outputs the following*

- *an integer $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$,*
- *a graph H and a family of sets $(A_x)_{x \in V(H)}$, $A_x \subseteq V(H)$, such that: $|V(H)| = t + o(t)$ and the graph H satisfies the following covering property: for each $x \in V(H)$ the set $A_x \subseteq V(H)$ is of size ℓ, $x \in A_x$, and $H[A_x] \in \Pi$.*

Furthermore, there exists a computation path where the H outputted above satisfies an additional property that H does not contain an induced subgraph of size $\ell + 1$ that belongs to Π.

Proof. We make a nondeterministic guess of a positive integer $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$ and a graph H_0 on $t^{\mathcal{O}(1)}$ vertices. By Lemma 3 there is an $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$ such that $R_\Pi(\ell+1) > R_\Pi(\ell) + t$. It can be easily verified that for the smallest such choice of ℓ we have $R_\Pi(\ell) + t = t^{\mathcal{O}(1)}$. Hence, in at least one computation path, H_0 is a graph on $R_\Pi(\ell) + t < R_\Pi(\ell+1)$ vertices which does not contain any induced subgraph from Π of size $\ell + 1$ (by the definition of $R_\Pi(\ell+1)$).

Then we cut the graph H from H_0: start with $S = \emptyset$ and, while $|S| < t$, repeatedly guess a set $A \subseteq V(H_0 \setminus S)$ such that $H_0[A] \in \Pi$ and $|A| = \ell$ and take $S := S \cup A$. At each step, we verify whether $H_0[A] \in \Pi$: if not, we terminate and output `FAIL`. Note that if H_0 has (at least) $R_\Pi(\ell) + t$ vertices then, as long as $|S| < t$, we have $|V(H_0 \setminus S)| \geq R_\Pi(\ell)$ and there exists at least one set A of size ℓ such that $H_0[A] \in \Pi$. This guarantees that feasible sets A are found on at least one computation path. Finally, when $|S| \geq t$ we take $H = H_0[S]$; note that $t \leq |S| < t + \ell = t + o(t)$. □

Proof (of Lemma 2). Let $I_1, \ldots, I_t$ be t instances of Improvement Π-Induced Subgraph of maximum size N. It is straightforward to efficiently partition such instances into $N^{\mathcal{O}(1)}$ equivalence classes: a) malformed instances, b) instances which are trivially NO since k exceeds the number of vertices (which is bounded by N), c) remaining instances with equal value of k. Note that all triples (G, X, k) with $G[X] \notin \Pi$ are treated as malformed instances and can be sorted into the first equivalence class since Π is polynomially recognizable. Hence we may assume well-formed instances of the form $I_j = (G_j, X_j, k)$, with $k \leq N$ and $G_j[X_j] \in \Pi$, since compositions for classes a) and b) are trivial.

By possibly duplicating some instances, we may assume that t is larger than the constant $\delta(\Pi)$ given by Lemma 3 for the class Π.

We start by invoking the algorithm from Lemma 4 for the class Π and integer t, but modify its outputs appropriately. If the algorithm would return `FAIL`, we output a trivial YES-instance of Π-Induced Subgraph instead.

Otherwise, we use the graph H and the integer ℓ to construct a graph G' by embedding one graph G_i into each vertex of H; as $|V(H)| = t + o(t)$, each graph G_i is embedded at least once. More formally, we take an arbitrary surjective function $\sigma : V(H) \to \{1, 2, \ldots, t\}$ and let $G' = \mathrm{Embed}(H; (G_{\sigma(x)})_{x \in V(H)})$. Set $k' = \ell(k-1)+1$. We return the instance (G', k') of Π-Induced Subgraph. Clearly, as $\ell = \mathcal{O}(\log^{1/\varepsilon} t)$, we have $k' \leq (N + \log t)^{\mathcal{O}(1)}$ and the algorithm runs in nondeterministic polynomial time. We now verify its correctness.

Assume first that (G_i, X_i, k) is a YES-instance to Improvement Π-Induced Subgraph for some $1 \leq i \leq t$: let $Y \subseteq V(G_i)$, $|Y| = k$, $G_i[Y] \in \Pi$. Recall that in this case the coNP-cross-composition algorithm should output a YES-instance in every computation path; this is clearly true if the algorithm of Lemma 4 fails to construct a graph H. Otherwise, let $x \in V(H)$ be such that $\sigma(x) = i$; recall that $x \in A_x \subseteq V(H)$, $|A_x| = \ell$, and $H[A_x] \in \Pi$. Let $Y_x = Y$ and $Y_y = X_{\sigma(y)}$ for $y \in A_x \setminus \{x\}$. Then $Y' = \{v(y, u) : y \in A_x, u \in Y_y\}$ is a set of size $k' = \ell(k-1)+1$ that induces in G' a graph isomorphic to $\mathrm{Embed}(H[A_x]; (G[Y_y])_{y \in A_x})$. As $H[A_x] \in \Pi$ and $G_{\sigma(y)}[Y_y] \in \Pi$ for $y \in A_x$, we infer that $G'[Y'] \in \Pi$ (since Π is closed under embedding) and (G', k') is a YES-instance for Π-Induced Subgraph.

For the second case, assume that no graph G_i contains an induced subgraph of size k that belongs to Π. Let us focus on a computation path where a graph H and integer ℓ are generated such that H does not contain an induced subgraph from Π of size $\ell + 1$. We claim that in this computation path the generated instance (G', k') is a NO-instance for Π-Induced Subgraph. Assume the contrary: let $Y' \subseteq V(G')$, $|Y'| = k'$, and $G'[Y'] \in \Pi$. Let $A = \{x \in V(H) : \exists_u : v(x, u) \in Y'\}$. Note that $H[A]$ is an induced subgraph of $G'[Y']$, thus $H[A] \in \Pi$. Since H does not contain an induced subgraph from Π of size $\ell + 1$, $|A| \leq \ell$. As $|Y'| = k' = \ell(k-1)+1$, by Pigeonhole Principle, there exists $x \in A$ such that $Y = \{u \in V(G_{\sigma(x)}) : v(x, u) \in Y'\}$ is of size at least k. Moreover, $G_{\sigma(x)}[Y]$ is an induced subgraph of $G'[Y'] \in \Pi$, thus Y induces a subgraph in $G_{\sigma(x)}$ of size at least k that belongs to Π, a contradiction. □

By a similar but slightly more technical coNP-cross-composition we can prove a similar result for graph classes that are not necessarily closed under embedding, but instead exclude a certain biclique $K_{s,s}$; e.g., chordal graphs, which exclude $C_4 = K_{2,2}$.

Theorem 6 (♠). *Unless NP $\subseteq$ coNP/poly, there does not exist a kernelization algorithm with polynomial guarantee on the output size for Π-*Induced Subgraph *for any non-trivial hereditary graph class Π that is polynomially recognizable, closed under disjoint union and excludes a certain biclique.*

The proof of Theorem 6 inspired the slightly more general lower bound result captured by Theorem 7 which is obtained by giving a polynomial parameter reduction from the Ramsey problem to Π-Induced Subgraph; this is showed in the following section, and Theorem 7 covers both split and chordal graphs as these classes forbid induced cycles of length 4 (i.e. $K_{2,2}$).

5 Polynomial Parameter Transformation from RAMSEY

In this section we establish the following theorem which rules out polynomial kernelizations for Π-Induced Subgraph when Π excludes some biclique $K_{s,s}$.

Theorem 7. *Unless NP $\subseteq$ coNP/poly, there does not exist a kernelization algorithm with polynomial guarantee on the output size for Π-*Induced Subgraph *for any non-trivial hereditary graph class Π that is polynomially recognizable, contains all independent sets, but excludes a certain biclique.*

We note that Theorem 7 is more general than Theorem 6, e.g., by including split graphs, but still does not cover all the cases of Theorem 5, e.g., the classes of perfect graphs and cographs are closed under embedding, but contain all bicliques.

We prove the theorem by a polynomial parameter transformation from the Ramsey problem. Let us recall the problem definition.

Ramsey **Parameter:** k.
Input: A graph G and an integer k.
Question: Does G contain an independent set or a clique of size k?

The following lemma, together with Theorem 3 and NP-hardness and kernelization hardness of Ramsey [8], proves Theorem 7. Note that polynomial recognizability of Π ensures us that Π-Induced Subgraph is in NP.

Lemma 5. *For any graph class Π that is polynomially recognizable, contains all independent sets, but excludes some biclique, there exists a polynomial parameter transformation that, given an instance (G, k) of* Ramsey*, outputs an equivalent instance (G', k') of Π-*Induced Subgraph.

Proof. Let $K_{s,s}$ be a biclique that is not in Π.

Let G^2 be the join of G and its complement $\overline{G}$; that is, $V(G^2) = V(G) \uplus V(\overline{G})$ and $E(G^2) = E(G) \cup E(\overline{G}) \cup \{uw : u \in V(G), w \in V(\overline{G})\}$. We define the graph G' as a graph obtained from G^2 by embedding an independent set of size ℓ into each vertex, where ℓ is a parameter, polynomially bounded in k, that will be chosen later. In other words, $G' = \mathrm{Embed}(G^2; (\overline{K_\ell})_{x \in V(G^2)})$. Let $k' = \ell k$. We claim that, if ℓ is sufficiently large, the Ramsey instance (G, k) is equivalent to the Π-Induced Subgraph instance (G', k').

In one direction, let (G, k) be a YES-instance to Ramsey. Then G^2 contains an independent set of size k, say $X \subseteq V(G^2)$. Then $X' = \{v(w, i) : w \in X, 1 \leq i \leq \ell\}$ is an independent set of size $k' = k\ell$ in G', and (G', k') is a YES-instance to Π-Induced Subgraph.

In the other direction, let $X' \subseteq V(G')$ be a set of size k' such that $G'[X'] \in \Pi$. Let $Y = \{w \in V(G^2) : \exists_i v(w, i) \in X'\}$ and $Z = \{w \in Y : |\{1 \leq i \leq \ell : v(w, i) \in X'\}| \geq s\}$. The key observation is that since $G'[X']$ does not contain the

Table 1. A partial catalogue of graph classes covered by our results; all except split graphs are closed under disjoint union

Graph Class	closed under Embedding	Has E-H Property	Forbids Biclique	Covered by Theorem
Perfect, Weakly Chordal, Cographs, Comparability	Yes	Yes	No	5
Chordal, Interval, Cluster, Proper Interval	No	Yes	Yes, $K_{2,2}$	6 and 7
Split	No	Yes	Yes, $K_{2,2}$	7
Claw-Free	No	Yes	Yes, $K_{3,3}$	6 and 7
AT-free	Yes	Open	No	5 (assuming E-H Conjecture)

biclique $K_{s,s}$ as a subgraph, Z is an independent set in G^2. Indeed, if $u, w \in Z$ and $uw \in E(G^2)$ then any s vertices of the form $v(w,i)$ together with any s vertices of the form $v(u,j)$ induce $K_{s,s}$ in G'. As G^2 is a join of G and $\overline{G}$, we infer that Z is wholly contained in $V(G)$ or wholly contained in $V(\overline{G})$.

If $|Z| \geq k$, we are done, as Z induces an independent set in G or in $\overline{G}$. Otherwise, as $|Z| < k$ but $|X'| = k'$, we infer that $|Y \setminus Z| > \ell/s$. By Lemma 1, there exists a constant $\varepsilon = \varepsilon(K_{s,s}) > 0$, such that for any $K_{s,s}$-free graph H (in particular, for any $H \in \Pi$), $\hom(H) \geq |V(H)|^{\varepsilon}$, and hence Π satisfies the Erdős-Hajnal property with the exponent ε. Let $\ell = s(2k)^{1/\varepsilon}$, so that $|Y \setminus Z| > (2k)^{1/\varepsilon}$. Note that for any fixed Π, ℓ and k' are polynomially bounded in k.

Now, $G^2[Y] \in \Pi$, as $G^2[Y]$ is isomorphic to an induced subgraph of $G'[X']$, and hence it contains a set X that induces a clique or an independent set of size $2k$ in G^2. At least k vertices of X lie in G or in $\overline{G}$, and the Ramsey instance (G, k) is a YES-instance. □

6 Conclusion

We showed how to use improvement versions of NP-hard problems for compositions without requiring them to be NP-hard, by using the much weaker property of NP-hardness under co-nondeterministic many-one reductions. We believe that this may simplify future lower bound proofs. Using this tool, we showed kernelization hardness results for Π-Induced Subgraph for polynomially recognizable hereditary graph classes that either exclude bicliques or are closed under embedding, that also satisfy the Erdős-Hajnal property. Table 1 contains a partial list of graph classes covered by our results of the paper. Note that if the kernelization hardness result holds for a class Π, then it also holds for the class that contains the complement of each of the graphs in Π.

It would be interesting to know whether there are any non-trivial hereditary properties Π for which Π-Induced Subgraph has a polynomial sized kernel. We conjecture that there are none.

References

1. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. Journal of Computer and System Sciences 75(8), 423–434 (2009)
2. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Cross-composition: A new technique for kernelization lower bounds. In: STACS, pp. 165–176 (2011)
3. Dell, H., Marx, D.: Kernelization of packing problems. In: SODA, pp. 68–81 (2012)
4. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: STOC, pp. 251–260 (2010)
5. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through Colors and IDs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
6. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. Journal of Computer and System Sciences 77, 91–106 (2011)
7. Hermelin, D., Wu, X.: Weak compositions and their applications to polynomial lower bounds for kernelization. In: SODA, pp. 104–113 (2012)
8. Kratsch, S.: Co-nondeterminism in compositions: a kernelization lower bound for a ramsey-type problem. In: SODA, pp. 114–122 (2012)
9. Harnik, D., Naor, M.: On the compressibility of NP instances and cryptographic applications. SIAM Journal on Computing 39(5), 1667–1713 (2010)
10. Khot, S., Raman, V.: Parameterized complexity of finding subgraphs with hereditary properties. Theoretical Computer Science 289, 997–1008 (2002)
11. Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)
12. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. Compositio Mathematica 2, 463–470 (1935)
13. Erdős, P.: Some remarks on the theory of graphs. Bulletin of the American Mathematical Society 53, 292–294 (1947)
14. Erdős, P., Hajnal, A.: Ramsey-type theorems. Discrete Applied Mathematics 25(1-2), 37–52 (1989)
15. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processing Letters 58, 171–176 (1996)
16. Marx, D.: Chordal deletion is fixed-parameter tractable. Algorithmica 57(4), 747–768 (2010)
17. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. Algorithmica 62(3-4), 807–822 (2012)
18. van Bevern, R., Komusiewicz, C., Moser, H., Niedermeier, R.: Measuring Indifference: Unit Interval Vertex Deletion. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 232–243. Springer, Heidelberg (2010)
19. Villanger, Y.: Proper Interval Vertex Deletion. In: Raman, V., Saurabh, S. (eds.) IPEC 2010. LNCS, vol. 6478, pp. 228–238. Springer, Heidelberg (2010)
20. Diestel, R.: Graph Theory. Springer (2005)
21. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 2nd edn. Elsevier Science (2004)
22. Alon, N., Pach, J., Solymosi, J.: Ramsey-type theorems with forbidden subgraphs. Combinatorica 21(2), 155–170 (2001)
23. Lovasz, L.: Perfect graphs. In: Beineke, L.W., Wilson, R.J. (eds.) Selected Topics in Graph Theory, vol. 2, pp. 55–67. Academic Press, London (1983)
24. Bodlaender, H.L., Thomassé, S., Yeo, A.: Kernel bounds for disjoint cycles and disjoint paths. Theoretical Computer Science 412(35), 4570–4578 (2011)

Testing Formula Satisfaction*

Eldar Fischer[1], Yonatan Goldhirsh[1], and Oded Lachish[2]

[1] Department of Computer Science, Technion, Haifa 32000, Israel
{eldar,jongold}@cs.technion.ac.il
[2] Birkbeck, University of London, London, UK
oded@dcs.bbk.ac.uk

Abstract. We study the query complexity of testing for properties defined by read once formulae, as instances of *massively parametrized properties*, and prove several testability and non-testability results. First we prove the testability of any property accepted by a Boolean read-once formula involving any bounded arity gates, with a number of queries exponential in ϵ and independent of all other parameters. When the gates are limited to being monotone, we prove that there is an *estimation* algorithm, that outputs an approximation of the distance of the input from satisfying the property. For formulae only involving And/Or gates, we provide a more efficient test whose query complexity is only quasipolynomial in ϵ. On the other hand we show that such testability results do not hold in general for formulae over non-Boolean alphabets; specifically we construct a property defined by a read-once arity 2 (non-Boolean) formula over alphabets of size 4, such that any 1/4-test for it requires a number of queries depending on the formula size.

1 Introduction

Property Testing deals with randomized approximation algorithms that operate under low information situations. The definition of a property testing algorithm uses the following components: A set of *objects*, usually the set of strings Σ^* over some alphabet Σ; a notion of a single *query* to the input object $w = (w_1, \dots, w_n) \in \Sigma^*$, which in our case would consist of either retrieving the length $|w|$ or the i'th letter w_i for any i specified by the algorithm; and finally a notion of *farness*, a normalized distance, which in our case will be the Hamming distance — $\texttt{farness}(w, v)$ is defined to be ∞ if $|w| \neq |v|$ and otherwise it is $|\{i : w_i \neq v_i\}|/|v|$.

Given a *property* P, that is a set of objects $P \subseteq \Sigma^*$, an integer q, and a farness parameter $\epsilon > 0$, an *ϵ-test for P with query complexity q* is an algorithm that is allowed access to an input object only through queries, and distinguishes between inputs that satisfy P and inputs that are ϵ-far from satisfying P (that is inputs whose farness from any object from P is more than ϵ) while using at most q queries. By their nature the only possible testing algorithms are probabilistic, with either 1-sided or 2-sided error (1-sided error algorithms must accept objects

* Research supported in part by an ERC-2007-StG grant number 202405.

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 376–387, 2012.

from P with probability 1). Traditionally the query "what is $|w|$" is not counted towards the q query limit.

The ultimate goal of Property-Testing research is to classify properties according to their optimal ϵ-test query-complexity. In particular, a property whose optimal query complexity depends on ϵ alone and not on the length $|w|$ is called *testable.* In many (but not all) cases a "query-efficient" property test will also be efficient in other computational resources, such as running time (usually it will be the time it takes to retrieve a query multiplied by some function of the number of queries) and space complexity (outside the space used to store the input itself).

Property-Testing was first addressed by Blum, Luby and Rubinfeld [4], and most of its general notions were first formulated by Rubinfeld and Sudan [19], where the investigated properties are mostly of an algebraic nature, such as the property of a Boolean function being linear. The first excursion to combinatorial properties and the formal definition of testability were by Goldreich, Goldwasser and Ron [12]. Since then Property-Testing has attracted significant attention leading to many results. For surveys see [6], [11], [17], [18].

Many times *families* of properties are investigated rather than individual properties, and one way to express such families is through the use of parameters. For example, k-colorability (as investigated in [12]) has an integer parameter, and the more general partition properties investigated there have the sequence of density constraints as parameters. In early investigations the parameters were considered "constant" with regards to the query complexity bounds, which were allowed to depend on them arbitrarily. However, later investigations involved properties whose "parameter" has in fact a description size comparable to the input itself. Probably the earliest example of this is [15], where properties accepted by a general read-once oblivious branching program are investigated. In such a setting a general dependency on the parameter is inadmissible, and indeed in [15] the dependency is only on the maximum width of the branching program, which may be thought of as a complexity parameter of the stated problem.

A fitting name for such families of properties is *massively parametrized properties.* A good way to formalize this setting is to consider an input to be divided to two parts. One part is the *parameter*, the branching program in the example above, to which the testing algorithm is allowed full access without counting queries. The other part is the *tested input*, to which the algorithm is allowed only a limited number of queries as above. Also, in the definition of farness only changes to the tested input are allowed, and not to the parameter. In other words, two "inputs" that differ on the parameter part are considered to be ∞-far. In this setting also other computational measures commonly come into play, such as the running time it takes to plan which queries will be made to the tested input.

Recently, a number of results concerning a massively parametrized setting (though at first not under this name) have appeared. See for example [13,5,8,10] and the survey [16], as well as [2], where such an ϵ-test was used as part of a larger mechanism.

A central area of research in Property-Testing in general and Massively-Parametrized Testing in particular is to associate the query complexity of problems to their other measures of complexity. There are a number of results in this direction, to name some examples see [1,15,9]. In [3] the study of formulae satisfiability was initiated. There it was shown that there exists a property that is defined by a 3-CNF formula and yet has a query complexity that is linear in the size of the input. This implies that knowing that a specific property is accepted by a 3-CNF formula does not give any information about its query complexity. In [14] it was shown that if a property is accepted by a read-twice CNF formula, then the property is testable. Here we continue this line of research.

In this paper we study the query complexity of properties that are accepted by read once formulae. These can be described as computational trees, with the tested input values at the leaves and logic gates at the other nodes, where for an input to be in the property a certain value must result when the calculation is concluded at the root.

We prove a number of results. Due to space considerations we provide in this extended abstract only the proofs of the most general of them, while the rest can be found in the full version [7]. Section 2 contains preliminaries. First we define the properties we test, and then we introduce numerous definitions and lemmas about bringing the formulas whose satisfaction is tested into a normalized "basic form". These are important and in fact implicitly form a preprocessing part of our algorithms. Once the formula is put in a basic form, testing an assignment to the formula becomes manageable.

In Section 3 we show the testability of properties defined by formulae involving arbitrary gates of bounded arity. We suppply a brief analysis of the running times of the algorithms in Section 4. One interesting implication of the testability results presented here, is that any read-once formula accepting an untestable Boolean property must use unbounded arity gates other than And/Or.

More results are available in the full version [7]: For such formula involving only monotone gates, we provide an *estimation* algorithm, that is an algorithm that not only tests for the property but with high probability outputs a real number η such that the true farness of the tested input from the property is between $\eta - \epsilon$ and $\eta + \epsilon$. We also show that when restricted to And/Or gates, we can provide a test whose query complexity is quasipolynomial in ϵ. On the other hand, we prove that these results can not be generalized to alphabets that have at least four different letters. We construct a formula utilizing only one (symmetric and binary) gate type over an alphabet of size 4, such that the resulting property requires a number of queries depending on the formula (and input) size for a 1/4-test.

2 Preliminaries

We use $[k]$ to denote the set $\{1, \ldots, k\}$. A *digraph* G is a pair (V, E) such that $E \subseteq V \times V$. For every $v \in V$ we set $\texttt{out-deg}(v) = \{u \in V \mid (u, v) \in E\}$. A *path* is a tuple $(u_1, \ldots, u_k) \in |V|^k$ such that $u_1, \ldots, u_k$ are all distinct and

$(u_i, u_{i+1}) \in E$ for every $i \in [k-1]$. The *length* of a path $(u_1, \dots, u_k) \in |V|^k$ is $k-1$. We say that there is a path from u to v if there exists a path $(u_1, \dots, u_k)$ in G such that $u_1 = u$, and $u_k = v$. The *distance* from $u \in V$ to $v \in V$, denoted $\texttt{dist}(u, v)$, is the length of the shortest path from u to v if one exists and infinity otherwise.

We use the standard terminology for outward-directed rooted trees. A *rooted directed tree* is a tuple (V, E, r), where (V, E) is a digraph, $r \in V$ and for every $v \in V$ there is an unique path from r to v. Let $u, v \in V$. If $\texttt{out-deg}(v) = 0$ then we call v a leaf. We say that u is an *ancestor* of v and v is a *descendant* of u if there is a path from u to v. We say that u is a *child* of v and v is a *parent* of u if $(v, u) \in E$, and set $\texttt{Children}(v) = \{w \in V \mid w \text{ is a child of } v\}$.

2.1 Formulae, Evaluations and Testing

With the terminology of rooted trees we now define our properties; first we define what is a formula and then we define what it means to satisfy one.

Definition 1 (Formula). *A* Read-Once Formula *is a tuple $\Phi = (V, E, r, X, \kappa, B, \Sigma)$, where (V, E, r) is a rooted directed tree, Σ is an alphabet, X is a set of variables (later on they will take values in Σ), $B \subseteq \bigcup_{k<\infty}\{\Sigma^k \mapsto \Sigma\}$ a set of functions over Σ, and $\kappa : V \to B \cup X \cup \Sigma$ satisfies the following (we abuse notation somewhat by writing κ_v for $\kappa(v)$).*

- *For every leaf $v \in V$ we have that $\kappa_v \in X \cup \Sigma$.*
- *For every v that is not a leaf $\kappa_v \in B$ is a function whose arity is $|\texttt{Children}(v)|$.*

In the case where B contains functions that are not symmetric, we additionally assume that for every $v \in V$ there is an ordering of $\texttt{Children}(v) = (u_1, \dots, u_k)$.

In the special case where Σ is the binary alphabet $\{0, 1\}$, we say that Φ is *Boolean.* Unless stated otherwise $\Sigma = \{0, 1\}$, in which case we shall omit Σ from the definition of formulae. A formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is called *read k-times* if for every $x \in X$ there are at most k vertices $v \in V$, where $\kappa_v \equiv x$. We call Φ a *read-once-formula* if it is read 1-times. A formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is called *k-ary* if the arity (number of children) of all its vertices is at most k. If a formula is 2-ary we then call it *binary.* A function $f : \{0, 1\}^n \to \{0, 1\}$ is *monotone* if whenever $x \in \{0, 1\}^n$ is such that $f(x) = 1$, then for every $y \in \{0, 1\}^n$ such that $x \leq y$ (coordinate-wise) we have $f(y = 1)$ as well. If all the functions in B are monotone then we say that Φ is (explicitly) *monotone.* We denote $|\Phi| = |X|$ and call is the *formula size.*

Definition 2 (Sub-Formula). *Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula and $u \in V$. The formula $\Phi_u = (V_u, E_u, u, X_u, \kappa, B)$, is such that $V_u \subseteq V$, with $v \in V_u$ if and only if* $\texttt{dist}(u, v)$ *is finite, and $(v, w) \in E_u$ if and only if $v, w \in V_u$ and $(v, w) \in E$. X_u is the set of all $\kappa_v \in X$ such that $v \in V_u$. If $u \neq r$ then we call Φ_u a* strict sub-formula. *We define $|\Phi_u|$ to be the number of variables in V_u, that is $|\Phi_u| = |X_u|$.*

Definition 3 (assignment to and evaluation of a formula). *An* assignment *σ to a formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is a mapping from X to Σ. The* evaluation *of Φ given σ, denoted (abusing notation somewhat) by $\sigma(\Phi)$, is defined as $\sigma(r)$ where $\sigma : V \to \Sigma$ is recursively defined as follows.*

- *If $\kappa_v \in \Sigma$ then $\sigma(v) = \kappa_v$.*
- *If $\kappa_v \in X$ then $\sigma(v) = \sigma(\kappa_v)$.*
- *Otherwise ($\kappa_v \in B$) we set $\sigma(v) = \kappa_v(\sigma(u_1), \ldots, \sigma(u_k))$, where* $\texttt{Children}(v) = (u_1, \ldots, u_k)$*.*

Given an assignment $\sigma : X \to \Sigma$ and $u \in V$, we let σ_u denote its restriction to X_u, but whenever there is no confusion we just use σ also for the restriction (as an assignment to Φ_u).

For Boolean formulae, we set $SAT(\Phi = b)$ to be all the assignments σ to Φ such that $\sigma(\Phi) = b$. When $b = 1$ and we do not consider the case $b = 0$ in that context, then we simply denote these assignments by $SAT(\Phi)$. If $\sigma \in SAT(\Phi)$ then we say that σ *satisfies* Φ. Let σ_1, σ_2 be assignments to Φ. We define $\texttt{farness}_\Phi(\sigma_1, \sigma_2)$ to be the relative Hamming distance between the two assignments. That is, $\texttt{farness}_\Phi(\sigma_1, \sigma_2) = |\{x \in X \mid \sigma_1(x) \neq \sigma_2(x)\}|/|\Phi|$. For every subset S of assignments to Φ we set $\texttt{farness}_\Phi(\sigma, S) = \min\{\texttt{farness}_\Phi(\sigma, \sigma') \mid \sigma' \in S\}$. If $\texttt{farness}_\Phi(\sigma, S) > \epsilon$ then σ is *ϵ-far* from S and otherwise it is *ϵ-close* to S.

We now have the ingredients to define testing of assignments to formulae in a massively parametrized model. Namely, the formula Φ is the parameter that is known to the algorithm in advance and may not change, while the assignment $\sigma : X \to \Sigma$ must be queried with as few queries as possible, and farness is measured with respect to the fraction of alterations it requires.

Definition 4. [**(ϵ, q)-test**] *An* (ϵ, q)-test *for $SAT(\Phi)$ is a randomized algorithm $\mathcal{A}$ with free access to Φ, that given oracle access to an assignment σ to Φ operates as follows.*

- *$\mathcal{A}$ makes at most q queries to σ (where on a query $x \in X$ it receives σ_x as the answer).*
- *If $\sigma \in SAT(\Phi)$, then $\mathcal{A}$ accepts (returns 1) with probability at least $2/3$.*
- *If σ is ϵ-far from $SAT(\Phi)$, then $\mathcal{A}$ rejects (returns 0) with probability at least $2/3$. Recall that σ is ϵ-far from $SAT(\Phi)$ if its relative Hamming distance from every assignment in $SAT(\Phi)$ is at least ϵ.*

We say that $\mathcal{A}$ is non-adaptive *if its choice of queries is independent of their values. We say that $\mathcal{A}$ has* 1-sided error *if given oracle access to $\sigma \in SAT(\Phi)$, it accepts (returns 1) with probability 1. We say that $\mathcal{A}$ is an* (ϵ, q)-estimator *if it returns a value η such that with probability at least $2/3$, σ is both $\eta + \epsilon$-close and $\eta - \epsilon$-far from $SAT(\Phi)$.*

We can now summarize the contributions of the paper in the following theorem. The first item is proved in this extended abstract, while the rest can be found in the full version [7]:

Theorem 5 (Main Theorem). *The following theorems all hold:*

- *For any read-once formula Φ where B is the set of all functions of arity at most k there exists a 1-sided (ϵ, q)-test for SAT(Φ) with $q = \exp(\text{poly}(\epsilon^{-1}))$.*
- *For any read-once formula Φ where B is the set of all monotone functions of arity at most k there exists an (ϵ, q)-estimator for SAT(Φ) with $q = \exp(\text{poly}(\epsilon^{-1}))$.*
- *For any read-once formula Φ where B is the set of all conjunctions and disjunctions of any arity there exists an (ϵ, q)-test for SAT(Φ) with $q = \epsilon^{O(\log \epsilon)}$.*
- *There exists an infinite family of 4 valued read-once formulae Φ such that there is no non-adaptive (ϵ, q)-test for SAT(Φ) with $q = O(\texttt{depth}(\Phi))$, and no adaptive (ϵ, q)-test for SAT(Φ) with $q = O(\log(\texttt{depth}(\Phi)))$.*

Note that for the first two items, the degree of the polynomial is linear in k.

2.2 Basic Formula Simplification and Handling

In the following, unless stated otherwise, our formulae will all be read-once and Boolean. For our algorithms to work, we will need a somewhat "canonical" form of such formulae. We say that two formulae Φ and Φ' are *equivalent* if $\sigma(\Phi) = \sigma(\Phi')$ for every assignment $\sigma : X \to \Sigma$.

Definition 6. *The* mDNF *(monotone disjunctive normal form) of a monotone boolean function $f : \{0,1\}^n \to \{0,1\}$ is a set of terms T where each term $T_i \in T$ is a subset $T_i \subseteq [n]$, there exists no two different terms $T_i, T_j \in T$ such that $T_i \subset T_j$, and for every $x \in \{0,1\}^n$, $f(x) = 1$ if and only if there exists a term $T_j \in T$ such that for all $i \in T_j$, we have that $x_i = 1$.*

Observation 7. *Any monotone boolean function $f : \{0,1\}^n \to \{0,1\}$ has a unique* mDNF *T.*

Definition 8. *For $u \in V$, $v \in \texttt{Children}(u)$ is called* (a,b)-forceful *if $\sigma(v) = a$ implies $\sigma(u) = b$. v is* forceful *if it is (a,b)-forceful for some $a, b \in \{0,1\}$.*

Forceful variables are variables that cause "Or-like" or "And-like" behavior in the gate.

Definition 9. *A vertex $v \in V$ is called* unforceable *if no child of v is forceful.*

Definition 10 (k-x-Basic formula). *A read-once formula Φ is k-x-*basic *if it is Boolean, all the functions in B have arity at least 2, and are either of arity at most k and unforceable, or $\wedge$ or $\vee$ of arity at least 2, and Φ satisfies the following. There is no $v \in V$ such that $\kappa_v \in \{0,1\}$. No $\wedge$ is a child of a $\wedge$ and no $\vee$ is a child of a $\vee$. Any variable may appear at most once in a leaf, either positively or negated.*

The set of variables that appear negated will be denoted by $\neg X$.

Lemma 11. *Every read-once formula Φ with gates of arity at most k has an equivalent k-x-basic formula Φ'.*

Proof. Suppose for some u that $v \in \texttt{Children}(u)$ is (a,b)-forceful. If $b = 1$ then κ_u can be replaced with an $\vee$ gate, where one input of the $\vee$ gate is v if $a = 1$ or the negation of v if $a = 0$, and the other input is the result of u when fixing $\sigma(\kappa_v) = 1 - a$. If $b = 0$ then κ_u can be replaced with an $\wedge$ gate, where one input of the $\wedge$ gate is v if $a = 0$ or the negation of v if $a = 1$, and the other input is the negation of the gate u when it is assumed that $\sigma(\kappa_v) = a$. After performing this transformation sufficiently many times we have no forceable gates left.

We will now eliminate $\neg$ gates. Any $\neg$ gate in the input or output of a gate which is not $\wedge$ or $\vee$ can be assimilated into the gate. Otherwise, a $\neg$ on the output of an $\vee$ can be replaced with an $\wedge$ with $\neg$'s on all of its inputs, according to De-Morgan's laws. Also by De-Morgan's laws, a $\neg$ on the output of a $\wedge$ can be replaced with an $\vee$ with $\neg$'s on all of its inputs.

Finally, any $\vee$ gates that have $\vee$ children can be merged with them, and the same goes for $\wedge$ gates. Now we have achieved an equivalent k-x-basic formula.

Note that $\vee$ and $\wedge$ gates are very much forceable.

2.3 Observations about Subformulae and Farness

Definition 12 (heaviest child $h(v)$). *Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula. For every $v \in V$ we define $h(v)$ to be v if $\texttt{Children}(v) = \emptyset$, and otherwise to be an arbitrarily selected vertex $u \in \texttt{Children}(v)$, such that $|\Phi_u| = \max\{|\Phi_w| \mid w \in \texttt{Children}(v)\}$.*

Definition 13 (vertex depth $\texttt{depth}_\Phi(v)$). *Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula. For every $v \in V$ we define $\texttt{depth}_\Phi(v) = \texttt{dist}(r, v)$ and $\texttt{depth}(\Phi) = \max\{\texttt{depth}_\Phi(u) \mid u \in V\}$.*

Observation 14. *Let $v \in V$ be such that either $\kappa_v \equiv \vee$ and $b = 0$ or $\kappa_v \equiv \wedge$ and $b = 1$, and $\texttt{farness}(\sigma, SAT(\Phi_v = b)) \geq \epsilon$. For every $1 > \alpha > 0$ there exists $S \subseteq \texttt{Children}(v)$ such that $\sum_{s \in S} |\Phi_s| \geq \epsilon\alpha^2|\Phi|$ and $\texttt{farness}(\sigma, SAT(\Phi_w = b)) \geq \epsilon(1 - \alpha)$ for every $w \in S$. Furthermore, the exists a child $u \in \texttt{Children}(v)$ such that $\texttt{farness}(\sigma, SAT(\Phi_u = b)) \geq \epsilon$.*

Proof. Let T be the maximum subset of $\texttt{Children}(v)$ such that Φ_w is $\epsilon(1 - \alpha)$-far from being evaluated to b for every $w \in T$. If $\sum_{t \in T} |\Phi_t| < \epsilon\alpha^2|\Phi|$ then the distance from having Φ_v evaluate to b is at most $\epsilon\alpha^2 + \epsilon(1 - \alpha)(1 - \alpha) < \epsilon$, which contradicts the assumption.

For the last part, note that if no such child exists then Φ_v is ϵ-close to being evaluated to b.

Observation 15. *Let $v \in V$ be such that either $\kappa_v \equiv \vee$ and $b = 1$ or $\kappa_v \equiv \wedge$ and $b = 0$, and $\texttt{farness}(\sigma, SAT(\Phi_v = b)) \geq \epsilon$. For every child $u \in \texttt{Children}(v)$, $|\Phi_u| \geq |\Phi|\epsilon$ and $\texttt{farness}(\sigma, SAT(\Phi_u = b)) \geq \epsilon(1 + \epsilon)$. Furthermore, $\epsilon \leq 1/2$, and for any $u \in \texttt{Children}(v) \setminus \{h(v)\}$, $\texttt{farness}(\sigma, SAT(\Phi_u = b)) \geq 2\epsilon$.*

Proof. First suppose that the weight of some child u is less than ϵ. In this case setting u to b makes the formula Φ_v evaluate to b by changing less than an ϵ fraction of inputs, a contradiction.

Since there are at least two children, every child u is of weight at most $1-\epsilon$ and since setting it to b would make Φ_v evaluate to b, it is at least $\epsilon(1+\epsilon)$-far from being evaluated to b.

For the last part, note that since Since $|\texttt{Children}(v)| > 1$, there exists $u \in \texttt{Children}(v)$ such that $|\Phi_u| \leq |\Phi_v|/2$. Thus every assignment to Φ_v is 1/2-close to an assignment σ' by which Φ_v evaluates to b. Also note that any $u \in \texttt{Children}(v) \setminus \{h(v)\}$ is of weight at most 1/2, and therefore if Φ_u were 2ϵ-close to being evaluated to b, Φ_v was ϵ-close to being evaluated to b.

2.4 Heavy and Light Children in General Gates

Definition 16. *Given a formula Φ, a parameter ϵ and a vertex u, we let $\ell = \ell(u, \epsilon)$ be the smallest integer such that the size of the ℓ'th largest child of u is less than $|\Phi|(4k/\epsilon)^{-\ell}$ if it exists, and set $\ell = k+1$ otherwise. The* heavy *children of u are the $\ell - 1$ largest children of u, and the rest of the children of u are its* light *children.*

Lemma 17. *If an unforceable vertex v has a child u such that $|\Phi_v|(1-\epsilon) \leq |\Phi_u|$, then σ is both ϵ-close to $SAT(\Phi_v = 1)$ and ϵ-close to $SAT(\Phi_v = 0)$.*

Observation 18. *If $\kappa_u \not\equiv \wedge$ and $\kappa_u \notin X$ and σ is ϵ-far from $SAT(\Phi_u = b)$, then it must have at least two heavy children.*

3 Upper Bound for General Bounded Arity Formula

Algorithm 1 tests whether the input is ϵ-close to having output b with 1-sided error, and also receives a confidence parameter δ. The explicit confidence parameter makes the inductive arguments easier and clearer.

Lemma 19. *The depth of recursion in Algorithm 1 is at most $16(4k/\epsilon)^k \log(\epsilon^{-1})$.*

Proof. If $\epsilon > 1$ then the condition in Line 1 is satisfied and the algorithm returns without making any queries.

All recursive calls occur in Lines 8, 13, 17 and 18.

Since Φ is k-x-basic, any call with a subformula whose root is labeled by $\wedge$ results in calls to subformulas, each with a root labeled either by $\vee$ or an unforceable gate, and with the same b value (this is crucial since the b value for which $\wedge$ recurses with a smaller ϵ is the b value for which $\vee$ recurses with a bigger ϵ, and vice-versa). Similarly, any call with a subformula whose root is labeled by $\vee$ results in calls to subformulas, each with a root labeled either by $\wedge$ or an unforceable gate, and with the same b value. Therefore, an increase of two in the depth results in an increase of the farness parameter from ϵ to at least

Algorithm 1

```
Input: read-once $k$-$x$-basic formula $\Phi = (V, E, r, X, \kappa)$, parameters $\epsilon, \delta > 0, b \in \{0, 1\}$,
oracle to $\sigma$.
Output: "true" or "false".
 1: if $\epsilon > 1$ then return "true"
 2: if $\kappa_r \in X$ then return the truth value of $\sigma(r) = b$
 3: if $\kappa_r \in \neg X$ then return the truth value of $\sigma(r) = 1 - b$
 4: if ($\kappa_r \equiv \wedge$ and $b = 1$) or ($\kappa_r \equiv \vee$ and $b = 0$) then
 5:     $y \longleftarrow$ "true"
 6:     for $i = 1$ to $l = 32(2k/\epsilon)^{2k} \log(\delta^{-1})$ do
 7:         $u \longleftarrow$ a vertex in Children($r$) selected independently at random, where the
            probability that $w \in$ Children($r$) is selected is $|\Phi_w|/|\Phi|$
 8:         $y \longleftarrow y \wedge$ Algorithm 1($\Phi_u, (\epsilon(1 - (2k/\epsilon)^{-k}/16)), \sigma, \delta/2, b$)
 9:     return $y$
10: if ($\kappa_r \equiv \wedge$ and $b = 0$) or ($\kappa_r \equiv \vee$ and $b = 1$) then
11:     if there exists a child of weight less than $\epsilon$ then return "true"
12:     $y \longleftarrow$ "false"
13:     for all $u \in$ Children($r$) do $y \longleftarrow y \vee$ Algorithm 1($\Phi_u, (\epsilon(1 + \epsilon)), \sigma, \epsilon\delta/2, b$)
14:     return $y$
15: if there is a child of weight at least $1 - \epsilon$ then return "true"
16: for all $u \in$ Children($r$) do
17:     $y_u^0 \longleftarrow$ Algorithm 1($\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 0$)
18:     $y_u^1 \longleftarrow$ Algorithm 1($\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 1$)
19: if There exists a string $x \in \{0, 1\}^k$ such that $\kappa_r$ on $x$ would evaluate to $b$ and for
    all $u \in$ Children($r$) we have $y_u^{x_u}$ equal to "true" then return "true" else return
    "false"
```

$(\epsilon(1 - (2k/\epsilon)^{-k}/16))(\epsilon(1 + (4k/\epsilon)^{-k})) \geq \epsilon(1 + (4k/\epsilon)^{-k}/16)$. Thus in recursive calls of depth $16(4k/\epsilon)^k \log(\epsilon^{-1})$ the farness parameter exceeds 1 and the call returns without making any further calls.

Lemma 20. *Algorithm 1 uses at most* $\epsilon^{-480(4k/\epsilon)^{k+3} \log\log(\delta^{-1})}$ *queries.*

The proof is standard and thus deferred to the full version. It follows from Lemma 19.

Lemma 21. *If Φ on σ evaluates to b then Algorithm 1 returns "true" with probability* 1*.*

The proof follows by an induction on formula depth and is deferred to the full version.

Lemma 22. *If σ is ϵ-far from getting Φ to output b then Algorithm 1 returns "false" with probability at least $1 - \delta$.*

Proof. The proof is by induction over the tree structure, where we partition to cases according to κ_r and b. Note that $\epsilon \leq 1$

If $\kappa_r \in X$ or $\kappa_r \in \neg X$ then by Lines 2 or 3 the algorithm returns "false" if σ is 0-far from getting Φ to output b.

If $\kappa_r \equiv \wedge$ and $b = 1$ or $\kappa_r \equiv \vee$ and $b = 0$, since σ is ϵ-far from getting Φ to output b then by Observation 14 we get that there exists $T \subseteq \texttt{Children}(r)$ such that $\sum_{t \in T} |\Phi_t| \geq |\Phi|\epsilon((2k/\epsilon)^{-2k}/16)$ and each Φ_t is $\epsilon(1-(2k/\epsilon)^{-k}/16)$-far from being evaluated to b. Let S be the set of all vertices selected in Line 7. The probability of a vertex from T being selected is at least $\epsilon((2k/\epsilon)^{-2k}/16)$. Since this happens at least $32(2k/\epsilon)^{2k}\log(\delta^{-1})$ times independently, with probability at least $1-\delta/2$ we have that $S \cap T \neq \emptyset$. Letting $w \in T \cap S$, the recursive call on it with parameter $\epsilon(1-(2k/\epsilon)^{-k}/16)$ will return "false" with probability at least $1-\delta/2$, which will evetually cause the returned value to be "false" as required. Thus the algorithm succeeds with probability at least $1-\delta$.

Now assume that $\kappa_r \equiv \wedge$ and $b = 0$ or $\kappa_r \equiv \vee$ and $b = 1$. Since Φ is ϵ-far from being evaluated to b, Observation 15 implies that all children are of weight at least ϵ, and therefore the conditions of Line 11 would not be triggered. Every recursive call on a vertex $v \in \texttt{Children}(r)$ is made with distance parameter $\epsilon(1+\epsilon)$ and so it returns "true" with probability at most $\epsilon\delta/2$. Since there are at most ϵ^{-1} children of r, the probability that none returns "true" is at least $1-\delta/2$ and in that case the algorithm returns "false" successfully.

Now assume that κ_r is some unforceable gate. By Observation 17, since Φ is ϵ-far from being satisfied the condition in Line 15 is not triggered. If the algorithm returned "true" then it must be that the condition in Line 19 is satisfied. If there exists some heavy $u \in \texttt{Children}(r)$ such that y_u^b is "true" and y_u^{1-b} is "false", then by Lemma 21 the formula Φ_u does evaluate to b and the string in x must be such that $x_u = b$. For the rest of the children of r, assuming the calls succeeded, each the subformula rooted in v is $(\epsilon(1+(4k/\epsilon)^{-k}))$-close to evaluate to x_v. Since u is heavy, the total weight of $\texttt{Children}(r) \setminus \{u\}$ is at most $1-(4k/\epsilon)^{-k}$, and thus by changing at most a $(\epsilon(1+(4k/\epsilon)^{-k}))(1-(4k/\epsilon)^{-k}) \leq \epsilon$ fraction of inputs we can get to an assignment where Φ evaluates to b.

If all heavy children u are such that both y_u^b and y_u^{1-b} are "true", then pick some heavy child u arbitrarily. Since r is unforceable, there is an assignment that evaluates to b no matter what the value of Φ_u is. Take such an assignment x that fits the real value of Φ_u. Note that for every heavy child v we have that $y_v^{x_v}$ is "true", and therefore by changing at most an $(\epsilon(1+(4k/\epsilon)^{-k}))$-fraction of the variables in Φ_v we can get it to evaluate to x_v. The weight of u is at least $(4k/\epsilon)^{-\ell+1}$, thus the total weight of the other heavy children is at most $1-(4k/\epsilon)^{-\ell+1}$ and the total weight of the light children is at most $\frac{\epsilon}{4}(4k/\epsilon)^{-\ell}$. So by changing all subformulas to evaluate to the value implied by x we change at most an $(\epsilon(1+(4k/\epsilon)^{-k}))(1-(4k/\epsilon)^{-\ell+1})+\frac{\epsilon}{4}(4k/\epsilon)^{-\ell} \leq \epsilon$ fraction of inputs and get an assignment where Φ evaluates to b. Note that this x is not necessarily the one found in Line 19.

Thus we have found that finding an assignment x in Line 19, assuming the calls are correct, implies that Φ is ϵ-close to evaluate to b. The probability that all relevant calls to an assignment return "true" incorrectly is at most the probability that the $2k$ recursive calls err, which by the union bound is at most δ, and the algorithm will return "false" correctly with probability at least $1-\delta$.

4 The Computational Complexity of the Testers and Estimator

There are two parts to analyzing the computational complexity of a test for a massively parametrized property. The first part is the running time of the preprocessing phase, which reads the entire parameter part of the input, in our case the formula, but has no access yet to the tested part, in our case the assignment. This part is subject to traditional running time and working space definitions, and ideally should have a running time that is quasi-linear or at least polynomial in the parameter size.

In our case, the preprocessing part would need to take a k-ary formula and convert it to the k-x-basic form corresponding to the algorithm that we run. We would also need to put the formula in a data structure that allows the following operations to be performed as quickly as possible:

For Algorithm 1, we would need to quickly pick a child of a vertex with probability proportional to its sub-formula size, and know who are the light children as well as what is the relative size of the smallest child. This mainly requires storing the size of every sub-formula for every vertex of the tree, as well as sorting the children of each vertex by their sizes and storing the value of the corresponding “ℓ”.

It is not hard to see that both constructing the normal form and calculating the above additional data can be done very efficiently. Furthermore, the only part that depends on epsilon is designating the light children, but this can also be done “for all epsilon” at a low cost (by storing the range of ϵ for every positive ℓ).

The second part is analyzing the running time complexity of the algorithm. Once the above preprocessing is performed, the time per instantiation (and thus per query) of the algorithm will be very small (where we charge the time it takes to calculate a recursive call to the recursive instantiation). This would make it a cost logarithmic in the input size per query (multiplied by the time it takes to write and read an address) – where the log incurrence is in fact only when we need to randomly choose a child according to its weight.

Acknowledgment. We thank Prajakta Nimbhorkar for the helpful discussion during the early stages of this work.

References

1. Alon, N., Krivelevich, M., Newman, I., Szegedy, M.: Regular languages are testable with a constant number of queries. SIAM J. Comput. 30(6), 1842–1862 (2000)
2. Ben-Sasson, E., Harsha, P., Lachish, O., Matsliah, A.: Sound 3-query pcpps are long. ACM Trans. Comput. Theory 1, 7:1–7:49 (2009)
3. Ben-Sasson, E., Harsha, P., Raskhodnikova, S.: Some 3CNF properties are hard to test. SIAM J. Comput. 35(1), 1–21 (2005)
4. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. J. Comput. Syst. Sci. 47(3), 549–595 (1993)

5. Chakraborty, S., Fischer, E., Lachish, O., Matsliah, A., Newman, I.: Testing *st*-Connectivity. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 380–394. Springer, Heidelberg (2007)
6. Fischer, E.: The art of uninformed decisions: A primer to property testing. Current Trends in Theoretical Computer Science: The Challenge of the New Century I, 229–264 (2004)
7. Fischer, E., Goldhirsh, Y., Lachish, O.: Testing formula satisfaction, arXiv:1204.3413v1 [cs.DS]
8. Fischer, E., Lachish, O., Newman, I., Matsliah, A., Yahalom, O.: On the query complexity of testing orientations for being eulerian. TALG (to appear)
9. Fischer, E., Newman, I., Sgall, J.: Functions that have read-twice constant width branching programs are not necessarily testable. Random Struct. Algorithms 24(2), 175–193 (2004)
10. Fischer, E., Yahalom, O.: Testing convexity properties of tree colorings. Algorithmica 60(4), 766–805 (2011)
11. Goldreich, O.: A Brief Introduction to Property Testing. In: Goldreich, O. (ed.) Property Testing. LNCS, vol. 6390, pp. 1–5. Springer, Heidelberg (2010)
12. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. J. ACM 45, 653–750 (1998)
13. Halevy, S., Lachish, O., Newman, I., Tsur, D.: Testing orientation properties. ECCC (153) (2005)
14. Halevy, S., Lachish, O., Newman, I., Tsur, D.: Testing properties of constraint-graphs. In: IEEE Conference on Computational Complexity (2007)
15. Newman, I.: Testing membership in languages that have small width branching programs. SIAM J. Comput. 31(5), 1557–1570 (2002)
16. Newman, I.: Property Testing of Massively Parametrized Problems - A survey. In: Goldreich, O. (ed.) Property Testing. LNCS, vol. 6390, pp. 142–157. Springer, Heidelberg (2010)
17. Ron, D.: Property testing: A learning theory perspective. Found. Trends Mach. Learn. 1, 307–402 (2008)
18. Ron, D.: Algorithmic and Analysis Techniques in Property Testing (2010)
19. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. 25(2), 252–271 (1996)

Reconstructing Strings from Substrings with Quantum Queries

Richard Cleve[1], Kazuo Iwama[2], François Le Gall[3], Harumichi Nishimura[4], Seiichiro Tani[5], Junichi Teruyama[2], and Shigeru Yamashita[6]

[1] Institute for Quantum Computing and School of Computer Science,
University of Waterloo, Canada
Perimeter Institute for Theoretical Physics, Canada
cleve@cs.uwaterloo.ca
[2] School of Informatics, Kyoto University, Japan
{fiwama,teruyamag}@kuis.kyoto-u.ac.jp
[3] Department of Computer Science, The University of Tokyo, Japan
legall@is.s.u-tokyo.ac.jp
[4] School of Information Science, Nagoya University, Japan
hnishimura@is.nagoya-u.ac.jp
[5] NTT Communication Science Laboratories, NTT Corporation, Atsugi, Japan
tani.seiichiro@lab.ntt.co.jp
[6] College of Information Science and Engineering, Ritsumeikan University, Japan
ger@cs.ritsumei.ac.jp

Abstract. This paper investigates the number of quantum queries made to solve the problem of reconstructing an unknown string from its substrings in a certain query model. More concretely, the goal of the problem is to identify an unknown string S by making queries of the following form: "Is s a substring of S?", where s is a query string over the given alphabet. The number of queries required to identify the string S is the query complexity of this problem.

First we show a quantum algorithm that exactly identifies the string S with at most $\frac{3}{4}N + o(N)$ queries, where N is the length of S. This contrasts sharply with the classical query complexity N. Our algorithm uses Skiena and Sundaram's classical algorithm and the Grover search as subroutines. To make them effectively work, we develop another subroutine that finds a string appearing only once in S, which may have an independent interest. We also prove two lower bounds. The first one is a general lower bound of $\Omega(\frac{N}{\log^2 N})$, which means we cannot achieve a query complexity of $O(N^{1-\epsilon})$ for any constant ϵ. The other one claims that if we cannot use queries of length roughly between $\log N$ and $3 \log N$, then we cannot achieve a query complexity of any sublinear function in N.

Keywords: quantum computing, string algorithms, query complexity, lower bounds.

1 Introduction

For an input of length N, we usually assume that the time complexity of any algorithm $\mathcal{A}$ is at least N, since $\mathcal{A}$ needs N steps only to read the input. However,

F.V. Fomin and P. Kaski (Eds.): SWAT 2012, LNCS 7357, pp. 388–397, 2012.

especially recently, there have been increasing demands for studying algorithms that run in significantly less than N steps by sacrificing the exactness of the computation. In this case, we obviously need some mechanism for algorithms to obtain the input, since it is no longer possible to read all the input bits sequentially. *Oracles* are a popular model for this purpose. The most standard oracle is so-called an *index oracle*, a mapping f from $\{0,1,\ldots,N-1\}$ into $\{0,1\}$ such that $f(i)$ returns the ith bit of the input. Thus, we need N oracle calls in order to get all the input bits. A little surprisingly, however, some Boolean functions can be computed, with high success probability, using oracle calls much less than N times. For example, a balanced AND-OR tree can be computed with $O(N^{0.753\ldots})$ oracle calls with high success probability [24].

This interesting fact becomes even more impressive if we are allowed to use *quantum* oracles. Due to the famous Grover search [16], we need only $O(\sqrt{N})$ oracle calls to compute the Boolean-OR function with high success probability, or a quadratic speed-up against its classical version (classically we need $\Omega(N)$ calls). This result is widely known as one of the two most remarkable examples claiming the superiority of quantum computation over classical computation (the other is Shor's integer factorization algorithm [23]).

To compute the Boolean-OR, it suffices to find at least one true value in the input bits. The *oracle identification problem*, or the *string reconstruction problem*, is more general and more difficult, namely it requires us to recover all the N bits of the input (thus any Boolean function can be computed without any additional oracle calls). The quantum index oracle is still nontrivially powerful for this problem; Ref. [9] shows that $N/2+O(\sqrt{N})$ oracle calls are enough for this problem, while we obviously need N queries in the classical counterpart. There are different types of oracles that are much more powerful for this most general problem. The quantum *IP oracle* [6], a function g from $\{0,1\}^N$ into $\{0,1\}$ such that $g(q)=q\cdot x$ for the input string x, needs only one oracle call to recover x while its classical counterpart N oracle calls. Recently, Ref. [18] studied the *balance oracle*, which models the balance scale to be used for the counterfeit coin problem (i.e., for finding the k counterfeit coins in N coins), and shows its quantum version can be solved with $O(k^{1/4})$ oracle calls while the classical version requires $\Omega(k\log(N/k))$ calls, where k is the number of 1's in x.

In 1993, Skiena and Sundaram [22] showed that $N+\Theta(\sqrt{N})$ (classical) queries are sufficient to reconstruct the hidden string x if we use a *substring oracle* or an *S-oracle*, in short. This oracle, $h(q)$, which returns 1 if the query string q is a substring of x, and 0 otherwise, had been quite popular in the algorithm community. For example it plays an important role in computational biology such as sequencing by hybridization [10,19,20]. One should notice that there is no obvious way (even regardless of its efficiency) of using this oracle for string reconstruction ($h(q)$ probably returns yes almost always if $|q|$ (the length of q) is short, say two or three, and no almost always if $|q|$ is, say, 10). Thus Skiena and Sundaram's result was highly appreciated, whose basic idea is as follows: Suppose that we already know that a substring s exists in the input x. Then we ask the oracle if $s1$ is a substring. If the answer is yes, we can increase the length of a confirmed substring by one. Otherwise, we know $s0$ is a substring or s is at the right end of x. Just assume the former and check the latter occasionally and we can get the above bound. It is almost tight information-theoretically.

Now here is our question in this paper: Is quantum also more powerful than classical computation for this oracle, and how much is it if yes? One might say the answer is easy: Instead of asking if $s1$ is a substring, we ask which of $x00$, $x01$, $x10$ and $x11$ is a substring using the 1/4-Grover search [7]. Since 1/4-Grover needs just one query, we can increase the confirmed substring by two per call, or we would get a roughly $N/2$ upper bound. Unfortunately it immediately turns out that this does not work, since more than one of the four candidates may be (correct) substrings of x at the same time (recall that 1/4-Grover only works for a unique solution).

Our Contribution. Here is our main result in this paper:

Theorem 1. *The quantum query complexity for identifying S-oracle is at most* $\frac{3}{4}N + O(\sqrt{N}\log N)$.

Therefore, the quantum algorithm is better than its classical counterpart by a factor of 3/4. Notice that our algorithm is *exact* as well as the classical one in [22]. To cope with the difficulty mentioned above, we use Skiena and Sundaram's algorithm until the confirmed substring gets to a certain length, then change our algorithm to the one based on 1/4-Grover. There still exists the possibility of multiple solutions, say $s00$ and $s01$, but now we can assume that s is pretty long or those two strings need to overlap if they are both solutions. This gives us a lot of information about the string s, which basically changes the problem into a certain kind of string manipulation problem that has a long history in theoretical computer science. By using this information, we construct the procedure which makes the situation that 1/4-Grover is useful.

Our strong conjecture is that our problem needs at least a linear number of queries. Our basic idea is to use the quantum adversary method [3,26], but it turns out that the fact that there is a wide range (one to N) in the length of query strings makes its direct application hard. We bypass this difficulty with two different approaches: The first one is to introduce a new query model, an *anchored substring* oracle, which is something between our substring oracle and the standard index oracle and makes it possible to exploit the basic ideas of the adversary method for the latter. This gives us the following theorem.

Theorem 2. *The quantum query complexity for identifying an S-oracle is* $\Omega\left(\frac{N}{\log^2 N}\right)$.

The second one is to prohibit a small range of length for available queries.

Theorem 3. *Suppose that we cannot use queries of length* $\log N - 1 - 2\log\log N$ *to* $3\log N$. *Then the problem of identifying an S-oracle needs* $\Omega(N)$ *queries.*

This theorem says that we need to use queries of the range of length between $\log N - 1 - 2\log\log N$ and $3\log N$ "effectively" to achieve a sublinear bound.

Related Work. There have been many studies achieving quantum linear speedups. As mentioned already, a most celebrated one is due to van Dam [9], who presented a quantum algorithm for identifying the oracle by $\frac{N}{2} + O(\sqrt{N})$ queries. This is optimal up to a constant factor since the lower bound $\frac{N}{4}$ was obtained by Ambainis [1]. Another example is ordered search, that is, to find a target in a sorted list of N items. Farhi et al. [15] invented a quantum algorithm that makes at most $c \log N$ queries with $c \approx 0.53$ (note that any classical algorithm needs at least $\log N$ queries), and the constant c was subsequently improved [11,5]. These linear speedups were also turned out to be tight (up to a constant factor) by the lower bound results in [2,17,12] which improved the previous lower bounds of [8,14].

There are no quantum studies based on substring oracles, and few ones about string manipulation previously. One of them is a quantum algorithm given by Ramesh and Vinay [21] which determines if a given pattern appears in a given text by combining Grover's search with a classical string matching technique called deterministic sampling.

Notes. The proofs of Theorems 2 and 3 are omitted here due to space constraints. They can be found in the full version of the present paper [13].

2 Upper Bounds

Now we give the definition of our oracle model. We call it a *substring oracle*, or simply an *S-oracle.*

Definition 1. *A substring oracle, or an S-oracle, in short, is a binary string* $x = x_0 \cdots x_{N-1} \in \{0,1\}^N$. *A query to an S-oracle is given as a string* $s \in \bigcup_{k=1}^{N} \{0,1\}^k$. *The answer from the S-oracle is a binary value* $\chi(x;s)$ *defined as follows: If* x *has* s *as substring, that is, there exists an integer* i *such that* $x_{i+k-1} = s_k$ *for all* $1 \le k \le |s|$ *then* $\chi(x;s) = 1$ *and otherwise* $\chi(x;s) = 0$. *In the quantum computation an S-oracle is viewed as the unitary transformation* $O_{S,x}$ *that transforms* $|s\rangle|a\rangle$ *to* $|s\rangle|a \oplus \chi(x;s)\rangle$.

To give the proof of Theorem 1, we define some notations on strings. The string representing the concatenation of strings u and v will be denoted uv. When $z = uv$, we call u a *prefix* of z and call v a *suffix* of z. A string v is called a *presuffix* of a string w if v is a prefix of w and also a suffix of w. The string formed by concatenating i copies of z will be denoted z^i. A string t is called the *periodic string* of a string a if t is the shortest string such that $a_i = t_{(i \mod |t|)}$ for all i (or, equivalently, t is the shortest string such that a can be written as $a = t^k b$ for some integer k and some prefix b of t).

2.1 Basic Ideas and Algorithms

Before the full description of our algorithm, we present the basic idea. The algorithm has three main steps. At the first step, we use Skiena and Sundaram's algorithm [22], which extends a substring in the oracle string x by one letter with one query. At the second step, we extend the substring z obtained by the first step to a string z_{out} so that z_{out} can appear only once in x. Note that the first and second steps are implemented classically. The third step is quantum: we apply Grover's search algorithm [16] under the special case that is called 1/4-*Grover search* [7]. Recall that the 1/4-Grover search can find a solution surely with only one query in the case when we know there is only one solution out of four candidates. Since the second step assures that the substring z_{out} appears only once in x, there is exactly one substring of x in $\{00z, 01z, 10z, 11z\}$ for any string z that extends z_{out} unless z corresponds to the leftmost part of x. So the 1/4-Grover search can extend the substring by two letters with only one query. If we know that z is a prefix of the oracle string, we run the 1/4-Grover search for $\{z00, z01, z10, z11\}$.

The second step is the most technical and it is also essential to implement the third step successfully. A key idea for obtaining the substring appearing only once is relatively simple; extending z by its periodic string. For instance, we assume $x = 10101101101101111110$. Then the substring $z = 1011011$ appears three times in x. The periodic string t of z is $t = 101$. Let us extend z by t as long as possible such that $t^i z$ is still a substring of x. In the example we get a substring $t^2 z = 1011011011011$, which appears only once in x. Now the difficulty is to make the string z obtained by the first step as short as possible, which improves the complexity of the algorithm. Another key idea for this difficulty is to analyze what happens when z appears twice in x. When a substring z with length $> N/2$ appears twice in x, these occurrences of z must be partially overlapping, and x has a substring uvw such that $z = uv = vw$. A key property is that the overlapped string v is a presuffix of z. Using these key ideas we can construct the algorithm by starting from the substring z of length $> N/2$.

Now we give an exact algorithm *Identify* and its subroutine *MakeOnce*.

Algorithm. *Identify*
Input : an S-oracle $O_{S,x}$.
Output : the oracle string x.
Step 1. Find a substring z of length $\lceil N/2 \rceil + 1$ using Skiena and Sundaram's algorithm [22].
Step 2. Run the algorithm *MakeOnce* on input z. Let z_{out} be the output.
Step 3. Repeat extending z_{out} to the left by 2 letters using the 1/4-Grover search. Check whether the extended string is a substring of x after every $\sqrt{N}$ applications of the 1/4-Grover search. If not, we know that a prefix of x is obtained between the current check point and the previous check point. Then, find this prefix by binary search.
Step 4. Repeat extending the current substring to the right by 2 letters using the 1/4-Grover search, and stop when the length of the substring becomes $N-1$ or N. If the length is $N-1$, use a classical query to find the last bit.
(End of Algorithm *Identify*)

Algorithm *MakeOnce*

```
Input : a string $z$ (a substring of $x$, $|z| > N/2$); an S-oracle $O_{S,x}$.
Output : a substring $z_{out}$ that appears only once in $x$.
Step 1. $T_0 := \emptyset$. $A_0 := \emptyset$. $l := 1$. $z_1 := z$. ($A_l$ is used for the analysis.)
Step 2. Repeat Steps 2.1–2.7.
  Step 2.1. Find the shortest string $a_l$ satisfying the following conditions.
    (i) $a_l$ is a presuffix of $z_l$.
    (ii) The periodic string of $a_l$ is not in $T_{l-1}$.
    If there is no such string, go to Step 3. Let $t_l$ be the periodic string of $a_l$.
    $T_l := T_{l-1} \cup \{t_l\}$. $A_l := A_{l-1} \cup \{a_l\}$.
  Step 2.2. Find the largest integer $i$ such that $(t_l)^i z_l$ is also a substring of $x$.
    Define $z'_l := (t_l)^i z_l$.
  Step 2.3. Let $j$ be the largest integer such that $z'_l = u t_l^j a_l$ for some string $u$.
  Step 2.4. Let $h$ be the largest integer such that $z'_l = t_l^h a_l w$ for some string $w$.
  Step 2.5. If $z'_l = t_l^j a_l$ or $h < j$, then $z_{l+1} := z'_l$ and go to Step 2.7.
  Step 2.6. Find the largest integer $k$ such that $u^k z'_l$ is also a substring of $x$. Define
    $z_{l+1} := u^k z'_l$.
  Step 2.7. $l := l + 1$.
Step 3. $l_{max} := l$. $z_{out} := z_{l_{max}}$. $T := T_{l_{max}-1}$. $A := A_{l_{max}-1}$. ($l_{max}$, $T$ and $A$
  are used for the analysis.)
(End of Algorithm MakeOnce)
```

2.2 Analysis of *MakeOnce*

In this section, we give the analysis of *MakeOnce*. First, a number of properties are given for the analysis.

Lemma 1. *For any $l < l_{max}$, MakeOnce satisfies the following properties.*

1. *$a_l \in A$ is represented as $a_l = t_l b_l$, where $t_l \in T$ and $|b_l| < |t_l|$.*
2. *z'_l and $a_l \in A$ are prefixes of z_{l+1}.*
3. *z_l (and hence $a_l \in A$) is a suffix of z_{l+1}.*
4. *a_l is a presuffix of a_{l+1} and $|a_{l+1}| > |a_l|$.*
5. *$|a_{l+1}| \geq |a_l| + |t_l|$.*
6. *$|t_{l+1}| > |t_l|$.*
7. *At step 2.6, $|u| > |t_l|$.*
8. *$l_{max} = O(\sqrt{N})$.*

Now we analyze the query complexity and the correctness of *MakeOnce*. In what follows, we refer to the properties 1–8 of Lemma 1 as simply the properties 1–8.

Proposition 1. *MakeOnce uses at most $O(\sqrt{N}\log N)$ queries.*

Proof. To obtain z_{out}, we need queries only at Step 2.2 and Step 2.6. These steps can be implemented by binary search to find $t_l^i z_l$ and $u^k z'_l$, which use $O(\log N)$ queries. Since the number of repetitions of Step 2 is $O(\sqrt{N})$ by the property 8, the total number of queries is $O(\sqrt{N}\log N)$. □

Proposition 2. *The output z_{out} of MakeOnce appears exactly once in x.*

Proposition 2 is proved by contradiction. We assume that z_{out} appears twice in x. Since $|z_{out}| > \frac{N}{2}$, x has a substring uvw such that $z_{out} = uv = vw$, where $|u| = |w| > 0$. Then we can see that v has the following special form.

Lemma 2. *$v = t_l^m a_l$ for some $l > 0$ and $m \geq 0$ where $t_l \in T$ and $a_l \in A$.*

Proof. First we should notice that z_{out} has no substring which satisfies the conditions at Step 2.1 since we go to Step 3 and z_{out} is output only when there is no string satisfying the conditions at Step 2.1. On the contrary, v is a presuffix of z_{out}, which means that v satisfies the condition (i) of Step 2.1. This implies that v does not satisfy the condition (ii) of Step 2.1. That is, the periodic string of v must be t_l in T for some l. Hence, it is represented as $v = t_l^{m'} y$ where $l > 0$, $m' > 0$, $t_l \in T$, and $|y| < |t_l|$.

For $a_l \in A$, let b_l be the string such that $a_l = t_l b_l$ and $|b_l| < |t_l|$ as guaranteed by the property 1. By the property 3, a_l is a suffix of z_{out}. Also, v is a suffix of z_{out}. Thus y has suffix b_l or b_l has suffix y. Now we show that $y = b_l$ by contradiction. Assuming $|y| < |b_l|$, it must hold that $t_l b_l = y' t_l y$ for some y' such that $|y'| < |t_l|$. Then the length of the periodic string of a_l is at most $|y'|$, which contradicts that t_l is the periodic string of a_l. Assuming $|y| > |b_l|$, $y' t_l b_l = t_l y$ for some y' such that $|y'| < |t_l|$. Then the length of the periodic string of a_l is at most $|y'|$, which also leads to a contradiction.

By the above arguments, v is represented as $v = t_l^{m'} b_l = t_l^m a_l$ where $m = m' - 1$. □

The main statement for the correctness of *MakeOnce* is now stated as follows. (In the rest of this section, we assume that u, w, u' and w' have positive length.)

Lemma 3. *For any $l \leq l_{max}$, any $c < l$ and $m \geq 0$, x has no substring $ut_c^m a_c w$ such that $z_l = ut_c^m a_c = t_c^m a_c w$, $t_c \in T_{l-1}$ and $a_c \in A_{l-1}$.*

Then, by the assumption that z_{out} $(= z_{l_{max}})$ appears twice in x, Lemma 2 implies that x has a substring $ut_l^m a_l w$ for some $0 < l \leq l_{max} - 1$ and $m \geq 0$, which contradicts Lemma 3. This completes the proof of Proposition 2.

What remains is the proof of Lemma 3. We prove the statement by induction on l. The case of $l = 1$ is easy. In this case, $T_0 = A_0 = \emptyset$, $z_1 = u = w$ and $|z_1| = |z| > \frac{N}{2}$. Hence x does not have a substring $uw = z_1 z_1$. Next we assume that the statement holds for l, and show that the statement holds for $l + 1$. For this purpose, we first show the following lemma:

Lemma 4. *If x has $u' t_{c'}^{m'} a_{c'} w'$ as a substring such that $z_{l+1} = u' t_{c'}^{m'} a_{c'} = t_{c'}^{m'} a_{c'} w'$ for some $c' < l$ and $m' \geq 0$, then x also has $ut_c^m a_c w$ such that $z_l = ut_c^m a_c = t_c^m a_c w$ for some $c \leq c'$ and $m \geq 0$.*

Proof. By the property 3, z_l is a suffix of z_{l+1}. By the assumption, z_{l+1} appears twice in x, and hence z_l also appears twice in x. Since $|z_l| > N/2$, x has a substring uvw with $z_l = uv = vw$. Let t be the periodic string of v. Then v is represented as $t^{m_v}b$ for some $m_v > 0$, where $|b| < |t|$ and b is a prefix of t. Note that $|t| \leq |t_{c'}|$ since v is a suffix of $t_{c'}^{m'}a_{c'}$.

Now we show that there is $c \leq c'$ such that $t = t_c$ and $b = b_c$, where b_c is the string such that $t_c b_c = a_c$ as guaranteed by the property 1. First we show $t \in T_l$, which means that $t = t_c$ for some $c \leq c'$ by $|t| \leq |t_{c'}|$ and property 6. For contradiction, we assume that $t \notin T_l$ (and hence $\notin T_{l-1}$). Note that since v satisfies the condition (i) of Step 2.1 for the l-th loop (i.e., v is a presuffix of z_l), tb also satisfies this condition. Then, by $t \notin T_{l-1}$, tb satisfies the conditions (i) and (ii) at Step 2.1. Since a_l is the shortest string satisfying the conditions (i) and (ii) at Step 2.1, $|tb| \geq |a_l|$. This means that a_l is a prefix of tb. Then we have $|t_l| \leq |t| \leq |t_{c'}|$ and $c' < l$. This contradicts the property 6. Second we show $b = b_c$. To this end, it suffices to show $|b| = |b_c|$ because both b and b_c are prefixes of $t = t_c$. Assume that $|b| < |b_c|$. Then $yt_cb = t_cb_c$ for some y such that $|y| < |t_c|$ since $tb = t_cb$ is a suffix of z_l and also, by property 3, $a_c = t_cb_c$ is a suffix of z_l. Then, the length of a periodic string of a_c is at most $|y|$, which contradicts that t_c is a periodic string of a_c. By a similar argument, we also have a contradiction assuming that $|b| > |b_c|$. Thus $|b| = |b_c|$.

We conclude that $tb = t_cb_c = a_c$, which completes the proof of Lemma 4. □

Lemma 4 and the induction hypothesis imply: For any $c < l$ and $m \geq 0$, x has no substring $ut_c^m a_c w$ such that $z_{l+1} = ut_c^m a_c = t_c^m a_c w$, $t_c \in T_l$ and $a_c \in A_l$. We now show another lemma.

Lemma 5. *For any $m \geq 0$, x has no substring $u't_l^m a_l w'$ such that $z_{l+1} = u't_l^m a_l = t_l^m a_l w'$.*

Proof. For contradiction, we assume that there is an $m \geq 0$ such that x has a substring $u't_l^m a_l w'$ satisfying $z_{l+1} = u't_l^m a_l = t_l^m a_l w'$. Then we lead to a contradiction for all the possible three cases at Step 2.5: (1) $z'_l = t_l^j a_l$; (2) $h < j$; (3) the other case.

In case (1), since $t_l^j a_l = z_{l+1} = u't_l^m a_l$, we have $m < j$ and $u' = t_l^{j-m}$. Then $u't_l^m a_l w' = t_l^{j-m} z'_l$ is a substring of x, which contradicts the maximality of i for $z'_l = t_l^i z_l$ at Step 2.2.

In case (2), $h < j$ and $z_{l+1} := z'_l = ut_l^j a_l$ for some u. Note that $m \leq h$ since h is taken as the largest integer such that $z'_l = t_l^h a_l w$ for some w at Step 2.4. Thus $j > m$ and hence $u't_l^m a_l w' = ut_l^{j-m} z'_l$. This implies that $ut_l^{j-m} z'_l$ and hence $t_l^{j-m} z'_l$ are included in x, which contradicts the maximality of i for $z'_l = t_l^i z_l$ at Step 2.2.

In case (3) where $h \geq j$, we take the largest integer k such that $u^k z'_l = u^{k+1} t_l^j a_l$ is a substring of x, and let $z_{l+1} := u^{k+1} t_l^j a_l$ at Step 2.6. Notice that u does not have suffix t_l and $|u| > |t_l|$ by the property 7. This implies that if $z_{l+1} = u^{k+1} t_l^j a_l$ has a suffix $t_l^{j'} a_l$ then $j' \leq j$. By the assumption, z_{l+1} has $t_l^m a_l$ as a suffix, which means $m \leq j$. Moreover, we can show that $m = j$: Since z'_l is a prefix of z_{l+1} by the property 2, there is a string w'' such that $z_{l+1} = z'_l w''$. Then x includes $u't_l^m a_l w' = u^{k+1} t_l^{j-m} z'_l w''$. However, Step 2.2 means that x does not have a $t_l z'_l$, which implies that $m = j$. Then x have a substring $(u^{k+1} t_l^j a_l) w' = u^{k+1}(t_l^m a_l w') = u^{k+1} z_{l+1} = u^{2k+1} z'_l$, which contradicts the maximality of k at Step 2.6. □

By the above two lemmas, it has been shown that for any $c < l+1$ and $m \geq 0$, x has no substring $ut_c^m a_c w$ such that $z_{l+1} = ut_c^m a_c = t_c^m a_c w$, $t_c \in T_l$ and $a_c \in A_l$. That is, the statement of Lemma 3 for case $l+1$ holds under the assumption that it holds for case l. Now the proof of Lemma 3 is completed.

2.3 Analysis of *Identify*

First, by following the basic idea described in Section 2.1, the correctness of *Identify* is easily verified. The output z_{out} of *MakeOnce* appears in x only once by Proposition 2. This guarantees that the 1/4-Grover search can extend z by two letters successfully in Steps 3 and 4 unless the current string reaches the left or right end. Moreover, the algorithm knows if the string reaches the ends by the regular checking in Step 3 or by the current length in Step 4.

Second, we analyze the number of queries used in *Identify*. At Step 1, we find a substring of length $\lceil \frac{N}{2} \rceil + 1$ by extending a string by one letter with one query. Then the number of queries at Step 1 is $\lceil \frac{N}{2} \rceil + 1$. At Step 2, the subroutine *MakeOnce* uses $O(\sqrt{N} \log N)$ queries by Proposition 1. At Steps 3 and 4, we extend a substring of length longer than $\frac{N}{2}$ by two letters with one query. Note that the number of checking whether it is a substring of x is $O(\sqrt{N})$. Thus the number of queries at Steps 3 and 4 is at most $N/4 + O(\sqrt{N})$. Therefore, the total number of queries is at most $\frac{3N}{4} + O(\sqrt{N} \log N)$.

Now the proof of Theorem 1 is completed.

3 Conclusion

Obvious future works are a (possible) improvement of the constant factor for the upper bound and a challenge to a linear lower bound (we strongly believe there are no sublinear algorithms). For the former, one possibility is to exploit a parity computation as was done in [9,18]. However, we do not have any indication that parity is substantially easier than reconstruction itself for substring oracles. For the latter we at least need to get rid of the reduction since we have already lost a $\log N$ factor by that. (See Section B.2 in [13].) Different approaches like the polynomial method [4] do exist as a possibility, but we have no idea on this direction, either, at this moment.

References

1. Ambainis, A.: A note on quantum black-box complexity of almost all Boolean functions. Inf. Process. Lett. 71(1), 5–7 (1999)
2. Ambainis, A.: A better lower bound for quantum algorithms searching an ordered list. In: Proc. 40th FOCS, pp. 352–357 (1999)
3. Ambainis, A.: Quantum lower bounds by quantum arguments. J. Comput. Syst. Sci. 64(4), 750–767 (2002)
4. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. J. ACM 48(4), 778–797 (2001)
5. Ben-Or, M., Hassidim, A.: The Bayesian learner is optimal for noisy binary search (and pretty good for quantum as well). In: Proc. 49th FOCS, pp. 221–230 (2008)

6. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM J. Comput. 26(5), 1411–1473 (1997)
7. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte Der Physik 46, 493–505 (1998)
8. Buhrman, H., de Wolf, R.: A lower bound for quantum search of an ordered list. Inf. Process. Lett. 70(5), 205–209 (1999)
9. van Dam, W.: Quantum oracle interrogation: Getting all information for almost half the price. In: Proc. 39th FOCS, pp. 362–367 (1998)
10. Dramanac, R., Crkvenjakov, R.: DNA sequencing by hybridization. Yugoslav Patent Application 570 (1987)
11. Childs, A.M., Landahl, A.J., Parrilo, P.A.: Improved quantum algorithms for the ordered search problem via semidefinite programming. Physical Review A 75(3), 032335 (2007)
12. Childs, A.M., Lee, T.: Optimal Quantum Adversary Lower Bounds for Ordered Search. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 869–880. Springer, Heidelberg (2008)
13. Cleve, R., Iwama, K., Le Gall, F., Nishimura, H., Tani, S., Teruyama, J., Yamashita, S.: Reconstructing Strings from Substrings with Quantum Queries. arXiv:1204.4691 (2012)
14. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: A limit on the speed of quantum computation for insertion into an ordered list, arXiv:quant-ph/9812057
15. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Invariant quantum algorithms for insertion into an ordered list, arXiv:quant-ph/9901059
16. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proc. 28th STOC, pp. 212–219 (1996)
17. Høyer, P., Neerbek, J., Shi, Y.: Quantum complexities of ordered searching, sorting, and element distinctness. Algorithmica 34(4), 429–448 (2002)
18. Iwama, K., Nishimura, H., Raymond, R., Teruyama, J.: Quantum Counterfeit Coin Problems. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 85–96. Springer, Heidelberg (2010)
19. Lysov, Y.P., Florent'ev, V.L., Khorlin, A.A., Khrapko, K., Shik, V.V., Mirzabekov, A.D.: DNA Sequencing by hybridization with oligonucleotides. A novel method. Dokl. Acad. Sci USSR 303, 1508–1511 (1988)
20. Pevzner, P.A., Lipshutz, R.J.: Towards DNA Sequencing Chips. In: Privara, I., Ružička, P., Rovan, B. (eds.) MFCS 1994. LNCS, vol. 841, pp. 143–158. Springer, Heidelberg (1994)
21. Ramesh, H., Vinay, V.: String matching in $\tilde{\mathrm{O}}(\sqrt{n}+\sqrt{m})$ quantum time. J. Discrete Algorithms 1(1), 103–110 (2003)
22. Skiena, S.S., Sundaram, G.: Reconstructing strings from substrings. J. Computational Biol. 2(2), 333–353 (1995)
23. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
24. Snir, M.: Lower bounds on probabilistic linear decision trees. Theoret. Comput. Sci. 38, 69–82 (1985)
25. Špalek, R., Szegedy, M.: All quantum adversary methods are equivalent. Theory of Computing 2(1), 1–18 (2006)
26. Zhang, S.: On the power of Ambainis lower bounds. Theoret. Comput. Sci. 339(2-3), 241–256 (2005)

Author Index